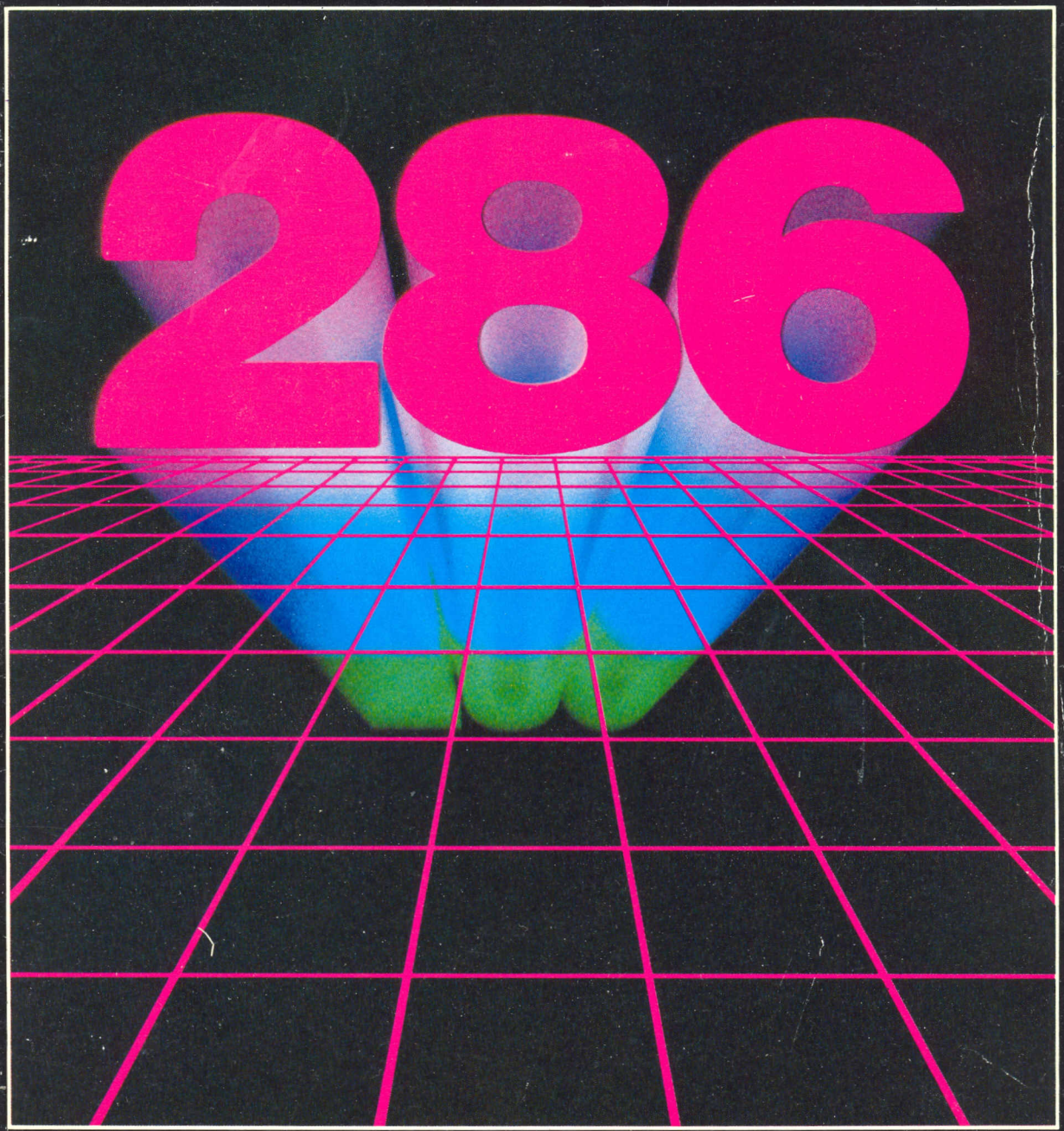
intel

iAPX 286
Hardware Reference Manual



Order Number 210760-001

LITERATURE

In addition to the product line Handbooks listed below, the INTEL PRODUCT GUIDE (no charge, Order No. 210846) provides an overview of Intel's complete product line and customer services.

Consult the INTEL LITERATURE GUIDE for a complete listing of Intel literature. TO ORDER literature in the United States, write or call the Intel Literature Department, 3065 Bowers Avenue, Santa Clara, CA 95051, (800) 538-1876, or (800) 672-1833 (California only). TO ORDER literature from international locations, contact the nearest Intel sales office or distributor (see listings in the back of most any Intel literature).

1984 HANDBOOKS	U.S. PRICE*
Memory Components Handbook (Order No. 210830) Contains all application notes, article reprints, data sheets, and other design information on RAMs, DRAMs, EPROMs, E^2PROMs, Bubble Memories.	**$15.00**
Telecommunication Products Handbook (Order No. 230730) Contains all application notes, article reprints, and data sheets for telecommunication products.	**7.50**
Microcontroller Handbook (Order No. 210918) Contains all application notes, article reprints, data sheets, and design information for the MCS-48, MCS-51 and MCS-96 families.	**15.00**
Microsystem Components Handbook (Order No. 230843) Contains application notes, article reprints, data sheets, technical papers for microprocessors and peripherals. (2 Volumes) (Individual User Manuals are also available on the 8085, 8086, 8088, 186, 286, etc. Consult the Literature Guide for prices and order numbers.)	**20.00**
Military Handbook (Order No. 210461) Contains complete data sheets for all military products. Information on Leadless Chip Carriers and on Quality Assurance is also included.	**10.00**
Development Systems Handbook (Order No. 210940) Contains data sheets on development systems and software, support options, and design kits.	**10.00**
OEM Systems Handbook (Order No. 210941) Contains all data sheets, application notes, and article reprints for OEM boards and systems.	**15.00**
Software Handbook (Order No. 230786) Contains all data sheets, applications notes, and article reprints available directly from Intel, as well as 3rd Party software.	**10.00**

*** Prices are for the U.S. only.**

intel

iAPX 286 HARDWARE REFERENCE MANUAL

1983

Intel Corporation makes no warranty for the use of its products and assumes no responsibility for any errors which may appear in this document nor does it make a commitment to update the information contained herein.

Intel retains the right to make changes to these specifications at any time, without notice.

Contact your local sales office to obtain the latest specifications before placing your order.

The following are trademarks of Intel Corporation and may only be used to identify Intel Products:

BXP, CREDIT, i, ICE, I^2ICE, ICS, iDBP, iDIS, iLBX, i_m, iMMX,
Insite, INTEL, int$_e$l, Intelevision, Intellec, int$_e$ligent Identifier™,
int$_e$lBOS, int$_e$ligent Programming™, Intellink, iOSP, iPDS,
iRMS, iSBC, iSBX, iSDM, iSXM, Library Manager, MCS,
Megachassis, Micromainframe, MULTIBUS, Multichannel™
Plug-A-Bubble, MULTIMODULE, PROMPT, Ripplemode,
RMX/80, RUPI, System 2000, and UPI, and the combination of
ICE, iCS, iRMX, iSBC, MCS, or UPI and a numerical suffix.

MDS is an ordering code only and is not used as a product name or trademark. MDS® is a registered trademark of Mohawk Data Sciences Corporation.

* MULTIBUS is a patented Intel bus.

Additional copies of this manual or other Intel literature may be obtained from:

Intel Corporation
Literature Department
3065 Bowers Avenue
Santa Clara, CA 95051

© INTEL CORPORATION, 1983

PREFACE

The Intel iAPX 286 microsystem is a high-performance microprocessing system based on the 80286 microprocessor.

This manual serves as the definitive hardware reference guide for iAPX 286 system designs. It is written for system engineers and hardware designers who understand the operating principles of microprocessors and microcomputer systems. Readers of this manual should already be familiar with the iAPX 286 architecture at the level described in the *Introduction to the iAPX 286* (Intel publication Order Number 210308).

In this manual, the iAPX 286 microsystem is presented from a hardware perspective. Information on the software architecture, instruction set, and programming of the iAPX 286 can be found in these related Intel publications:

- *iAPX 286 Programmer's Reference Manual*, Order Number 210498
- *ASM286 Assembly Language Reference Manual*, Order Number 121924
- *PL/M-286 User's Guide*, Order Number 121946

Together with the *iAPX 286 Hardware Reference Manual*, these publications provide a complete description of the iAPX 286 microsystem for hardware designers, software engineers, and all users of iAPX 286 systems.

ORGANIZATION OF THIS MANUAL

The information in this manual is divided into seven chapters and one appendix. The material is introduced beginning with a system-level description of the iAPX 286 microsystem, and continues with discussions of the detailed hardware design information needed to implement system designs using the actual iAPX 286 components.

- Chapter One provides an overview of the iAPX 286 microsystem.
- Chapter Two describes iAPX 286 system architecture from the viewpoint of the system designer. Examples illustrate the many different system configurations that are possible within this architecture. Design tradeoffs for the various choices are discussed.
- Chapter Three discusses the iAPX 286 local bus. Included in this chapter are detailed signal descriptions and timing for the 80286 and support components, a discussion of iAPX 286 memory and I/O organization, and in-depth discussions of processor and local bus interface guidelines.

The next four chapters provide the information required to interface memory, peripheral devices, and processor extensions to the 80286 microprocessor.

- Chapter Four discusses techniques for designing memory subsystems for the iAPX 286, and describes the effects of wait states on iAPX 286 performance.
- Chapter Five explains how to interface I/O devices to an iAPX 286 system.
- Chapter Six describes the interface between the 80286 and the 80287 Numeric Processor Extension, which greatly expands the mathematical processing power of the iAPX 286 microsystem.

- Chapter Seven shows how an iAPX 286 system can interface to the IEEE 796 Multibus, Intel's multi-master system bus.
- Appendix A contains data sheets for the iAPX 286/10 (80286) microprocessor, the 80287 Numeric Processor Extension, and the various processor support components.

Table of Contents

	Page
CHAPTER 1	
INTRODUCTION	
iAPX 286 Microsystem Components	1-1
Microprocessors	1-2
Interrupt Controller	1-2
Bus Interface Components	1-2
CHAPTER 2	
iAPX 286 SYSTEM ARCHITECTURE	
iAPX 286 Bus Organization	2-1
The iAPX 286 Local Bus	2-1
The Buffered Local Bus	2-2
The System Bus	2-3
Bus Interface Groups	2-3
Memory Subsystems for the iAPX 286	2-3
Local Memory	2-6
System Memory	2-7
Dual-Port Memory	2-7
I/O Subsystems for the iAPX 286	2-10
Processor Extensions	2-10
Multi-Processing Overview	2-13
Bus Arbitration	2-13
Mutual Exclusion	2-14
Using the iAPX 286 with the IEEE 796 MULTIBUS®	2-16
A Multi-Processing Design Example	2-16
CHAPTER 3	
THE iAPX 286 LOCAL BUS	
Introduction to the 80286 Central Processing Unit	3-1
Processor Architecture	3-2
The Bus Unit	3-2
The Instruction Unit	3-2
The Execution Unit	3-3
The Address Unit	3-3
The Effects of Pipelining	3-3
Operating Modes	3-4
The 80286 Bus Interface	3-4
Local Bus Overview	3-5
Organization of Physical Memory and I/O	3-5
Memory Organization	3-5
I/O Organization	3-9

Page

Interrupt Organization 3-10
Interrupt Priorities 3-11
Non-Maskable Interrupt Request (NMI) 3-12
Maskable Interrupt Request (INTR) 3-13
Pipelined Address Timing 3-15
iAPX 286 Local Bus States 3-16
iAPX 286 Bus Operations 3-17
Read Cycles 3-18
Write Cycles 3-18
Interrupt-Acknowledge Cycles 3-21
Halt/Shutdown Cycles 3-21
iAPX 286 Bus Usage 3-23
Local Bus Usage Priorities 3-23
Prefetch Operations 3-23
Data Operations 3-24
Data Channel Transfers 3-25
Bus Utilization 3-25
Bus Interface Components 3-26
Generating Timing Using the 82284 Clock Generator 3-26
Generating the System Clock 3-26
Timing Bus Operations Using $\overline{\text{READY}}$ 3-32
Wait-State Timing Logic 3-37
Generating the Proper RESET Timing 3-40
Synchronizing Processor Clocks in a Multi-Processor System 3-43
Controlling the Local Bus Using the 82288 Bus Controller 3-44
Systems Having More Than One Bus Controller 3-44
Selecting a Controller 3-45
Selecting MULTIBUS® Timing 3-46
Modifying the Bus Control Timing 3-48
CMDLY (Command Delay) 3-49
CEN/$\overline{\text{AEN}}$ (Command Enable/Address Enable) 3-49
Local Bus Design Considerations 3-52
Address Bus Interface 3-52
Address Decoding 3-53
Non-Latched Chip Selects 3-53
Latched Chip Selects 3-55
Data Bus Interface 3-56
Other Bus Masters on the iAPX 286 Local Bus 3-61
DMA Configuration 3-64
Initializing the iAPX 286 Processor 3-65
80286 Internal States 3-65
80286 External Signals 3-67
iAPX 286 Bus Timing 3-69
Address and ALE Timing 3-71

Page

Command Timing 3-71
READY Timing 3-72
Read Cycle Timing 3-72
Write Cycle Timing 3-73
Interrupt-Acknowledge Timing 3-74
Physical Design Considerations 3-74
Power, Ground, and Cap Connections 3-74
Debugging Considerations 3-76
The iLBX Bus—A High-Performance Local Bus Standard 3-76

CHAPTER 4
MEMORY INTERFACING

Memory Speed vs. Performance and Cost 4-1
System Performance and Memory Performance 4-2
Memory Speed and Memory Performance 4-2
iAPX 286 System Performance with Wait States 4-3
Explaining the Benchmark Results 4-3
The Intel Pascal Benchmarks 4-5
Queens 4-5
GCD 4-5
Bubble Sort 4-5
Matrix Multiply 4-5
The Intel Assembly-Language Benchmarks 4-6
Inspect 4-6
XLAT 4-6
BSORT 4-6
XFORM 4-6
PCALL 4-6
Memory Interface Techniques 4-6
Standard Address Strobe Logic 4-7
Special Address Strobe Logic 4-8
Interleaved Memory 4-9
Timing Analysis for Memory Operations 4-11
Standard ALE Timing 4-12
Read Operations 4-13
Write Operations 4-15
Special Address Strobe Timing 4-18
Interleaved Memory Timing 4-20
Memory Interface Examples 4-23
Read-Only Memory 4-23
Timing Analysis for ROMS 4-24
Static RAM Devices 4-25
Timing Analysis for Static RAMS 4-27
Pseudo-Static RAM Devices 4-30

Page

Dynamic RAM Devices ... 4-30
Timing Analysis for the 8207 RAM Controller ... 4-33
Error Correction Using the 8206 EDCU ... 4-38
Dual-Port Memory Subsystems Using the 8207 ADRC ... 4-39

CHAPTER 5
I/O INTERFACING

I/O-Mapping vs. Memory-Mapping ... 5-1
Address-Decoding ... 5-1
iAPX 286 Instruction Set ... 5-1
Device Protection ... 5-2
Interfacing to 8-Bit and 16-Bit I/O ... 5-3
Address Decoding ... 5-3
Memory-Mapped I/O ... 5-4
8-Bit I/O ... 5-5
16-Bit I/O ... 5-8
Linear Chip Selects ... 5-8
Timing Analysis for I/O Operations ... 5-8
Read Operations ... 5-10
Write Operations ... 5-12
Matching I/O Device Requirements ... 5-14
I/O Interface Examples ... 5-14
8274 Interface ... 5-15
Read Timing ... 5-16
Write Timing ... 5-17
8255A-5 Interface ... 5-17
8259A-2 Interface ... 5-18
Single Interrupt Controller ... 5-19
Cascaded Interrupt Controllers ... 5-19
Handling More than 64 Interrupts ... 5-20
The iSBX Bus—A Modular I/O Expansion Bus ... 5-21

CHAPTER 6
USING THE 80287 NUMERIC PROCESSOR EXTENSION

The 80287 Processor Extension Interface ... 6-1
The 80287 Status Lines ... 6-2
Addressing the 80287 ... 6-3
The 80287 Clock Input ... 6-3
Local Bus Activity with the 80287 ... 6-4
Execution of ESC Instructions ... 6-4
The Processor Extension Data Channel ... 6-4
Data Channel Requests ... 6-4

	Page
Data Channel Transfers	6-4
Data Channel Priority	6-6
Performance Using 80287 Parallel Processing	6-6
Designing an Upgradable iAPX 286 System	6-7
Designing the 80287 Socket	6-7
Recognizing the 80287	6-7
CHAPTER 7	
THE SYSTEM BUS	
The System-Bus Concept	7-1
The Division of Resources	7-2
Local Resources	7-2
System Resources	7-2
The IEEE 796 MULTIBUS®—A Multimaster System Bus	7-3
MULTIBUS® Design Considerations	7-3
Memory Operations	7-5
I/O Operations	7-6
Interrupt-Acknowledge to Cascaded Interrupt Controllers	7-7
Byte-Swapping During MULTIBUS® Byte Transfers	7-9
Implementing the Bus-Timeout Function	7-10
Power Failure Considerations	7-10
MULTIBUS® Arbitration Using the 82289 Bus Arbiter	7-12
Gaining Control of the MULTIBUS®	7-12
Releasing the Bus—Three 82289 Operating Modes	7-14
When to Use the Different Modes	7-15
Configuring the 82289 Operating Modes	7-16
Asserting the $\overline{\text{LOCK}}$ Signal	7-17
Timing Analysis of the MULTIBUS® Interface	7-18
Using Dual-Port RAM with the System Bus	7-19
Avoiding Deadlock with a Dual-Port Memory	7-19
APPENDIX A	
DEVICE SPECIFICATIONS	
iAPX 286/10 High Performance Microprocessor with Memory Management and Protection	A-1
80287 80-Bit HMOS Numeric Processor Extension	A-51
82284 Clock Generator and Ready Interface for iAPX 286 Processors	A-75
82288 Bus Controller for iAPX 286 Processors	A-83
8282/8283 Octal Latch	A-99
8286/8287 Octal Bus Transceiver	A-105
8207 Advanced Dynamic RAM (DRAM) Controller	A-111
8259A/8259A-2/8259A-8 Programmable Interrupt Controller	A-129

List of Tables

Table	Title	Page
1-1	iAPX 286 Microsystem Components	1-1
3-1	80286 Bus Cycle Status Decoding	3-4
3-2	Processing Order for Simultaneous Interrupts	3-11
3-3	Local Bus Usage Priorities	3-24
3-4	82284 Crystal Loading Capacitance Values	3-28
3-5	Bus Operations Affecting HOLD Latency	3-64
3-6	CPU State Following RESET	3-66
3-7	80286 Bus During RESET	3-68
3-8	82288 Command States During RESET	3-68
4-1	iAPX 286 Performance With Wait States	4-4
4-2	Comparing Several Memory Interface Techniques	4-12
4-3	Timing for 8-MHz Read Operations Using Standard ALE Strobe	4-14
4-4	Timing for 8-MHz Write Operations Using Standard ALE Strobe	4-17
4-5	Timing for 8-MHz Read Operations Using Special Address Strobe	4-20
4-6	Timing for 8-MHz Interleaved Memory Operations	4-22
4-7	iAPX 286 Performance with Interleaved Memories	4-23
4-8	Timing Analysis for the 2764-20 EPROM	4-25
4-9	Wait-State Requirements for Intel EPROMs	4-26
4-10	Timing Analysis for the 2147H-1 RAM	4-30
4-11	Non-ECC Mode Programming, PDI Pin (57) Tied to Ground	4-35
4-12	8-MHz Timing Analysis for the 2118-10 DRAM	4-37
4-13	6-MHz Timing Analysis for the 2164A-15 DRAM	4-37
5-1	Timing Analysis for 8-MHz I/O Read Operations	5-12
5-2	80286 Timing Analysis for I/O Write Operations	5-14
5-3	80286/Peripheral Timing Parameters	5-15
5-4	Timing Requirements for Selected Peripherals	5-15
6-1	I/O Address Decoding for the 80287	6-3
6-2	Whetstone Performance with Multiple Wait States	6-7
7-1	Local Bus and MULTIBUS® Usage of BHE/BHEN	7-10
7-2	Three 82289 Operating Modes for Releasing the Bus	7-15
7-3	Required MULTIBUS® Timing for the iAPX 286	7-18

List of Figures

Figure	Title	Page
2-1	Representative iAPX 286 System	2-2
2-2	iAPX 286 on Multi-Master System Bus	2-4
2-3	iAPX 286 With Both Local and System Buses	2-5
2-4	Representative iAPX 286 System Memory Map	2-6

Figure	Title	Page
2-5	Dual-Port Memory	2-8
2-6	Arbitration Logic for a Dual-Port Memory	2-8
2-7	8207 Dual-Port Subsystem	2-9
2-8	iAPX 286 With 8086/8089 IOP Subsystem	2-11
2-9	80286 With an 80287	2-12
2-10	80286 With DMA Controller	2-14
2-11	80286 on the MULTIBUS® Interface	2-15
2-12	iAPX 286 Design Example	2-17
3-1	80286 Internal Block Diagram	3-2
3-2	Operation of Sequential vs. Pipelined Processors	3-3
3-3	Separate Memory and I/O Spaces	3-6
3-4	iAPX 286 Memory Organization	3-6
3-5	80286 Byte Transfers	3-7
3-6	Even-Addressed Word Transfer	3-8
3-7	Odd-Addressed Word Transfer	3-9
3-8	Interrupt Descriptor Table Definition	3-10
3-9	NMI and INTR Input Timing	3-12
3-10	8259A PIC Driving 80286 INTR Input	3-14
3-11	Typical and Pipelined Bus Operations	3-15
3-12	System and Processor/PCLK Clock Relationships	3-16
3-13	80286 States	3-17
3-14	iAPX 286 Read Cycle	3-19
3-15	80286 Write Cycle	3-20
3-16	Interrupt-Acknowledge Sequence	3-22
3-17	82284 Clock Generator With an 80286 CPU	3-27
3-18	80286 Clock Input	3-27
3-19	Recommended Crystal Connections to the 82284	3-28
3-20	82284 With External Frequency Source	3-29
3-21	External Frequency for Multiple 82284s	3-30
3-22	Multiple Local Bus Processors Share a Common 82284	3-31
3-23	CLK to PCLK Timing Relationship	3-31
3-24	$\overline{\text{READY}}$ Timing	3-33
3-25	Ready Inputs to the 82284 and Output to the 80286 and 82288	3-33
3-26	$\overline{\text{SRDY}}$ and $\overline{\text{READY}}$ Timing	3-34
3-27	$\overline{\text{ARDY}}$ and $\overline{\text{READY}}$ Timing	3-35
3-28	Using Only One Ready Input	3-35
3-29	Selecting the Ready Input	3-36
3-30	Open-Collector 82284 $\overline{\text{READY}}$ Output	3-36
3-31	$\overline{\text{READY}}$ Output Characteristics	3-37
3-32	Generating a Single Wait State	3-38
3-33	Timing for the Single Wait-State Generator	3-38
3-34	Generating from 0 to 3 Wait States	3-39
3-35	Typical RC RESET Timing Circuit	3-41

Figure	Title	Page
3-36	80286 Reset and System Reset	3-41
3-37	Processor Clock Synchronization	3-42
3-38	RESET Synchronized to CLK	3-42
3-39	Generating a Synchronous RESET to Multiple 80286 CPUs	3-43
3-40	Synchronous RESET Circuit Timing	3-44
3-41	82288 Bus Controller with an 80286 CPU	3-45
3-42	CENL Disable of DEN on Write Cycle	3-46
3-43	Local (MB=0) Vs. MULTIBUS® (MB=1) Timing	3-47
3-44	Bus Controllers in Dual Bus System	3-48
3-45	Generating CMDLY For 0 or 1 CLK Delays	3-49
3-46	Generating 0 to 7 Command Delays	3-50
3-47	Bus Cycle Terminated with No Command Issued	3-50
3-48	CEN Characteristics (No Command Delay)	3-51
3-49	Latching the 80286 Address	3-52
3-50	ALE Timing	3-53
3-51	Dual Address Bus System	3-54
3-52	Non-Latched Chip Selects	3-54
3-53	Latched Chip Selects	3-55
3-54	Devices With Output Enables on a Non-Buffered Data Bus	3-56
3-55	Devices Without Output Enables on the Local Data Bus	3-57
3-56	Buffering the Data Bus	3-58
3-57	Dual Data Bus System	3-58
3-58	Double Buffered System	3-59
3-59	Controlling System Transceivers with DEN and DT/$\overline{R}$	3-60
3-60	Buffering Devices with $\overline{OE}$/RD	3-60
3-61	Buffering Devices without $\overline{OE}$/RD and with Common or Separate Input/Output	3-60
3-62	Buffering Devices without $\overline{OE}$/RD and with Separate Input/Output	3-61
3-63	Entering the Hold State	3-62
3-64	Exiting the Hold State	3-63
3-65	HOLD Input Signal Timing	3-64
3-66	DMA Controller on the Local Bus	3-65
3-67	DMA Controller on a Private System Bus	3-66
3-68	Decoding Addresses in Both Real and Protected Modes	3-67
3-69	Signal States During RESET	3-68
3-70	Disabling the 80286 Bus Interface on RESET	3-69
3-71	iAPX 286 Local Bus Cycle Timing	3-70
3-72	80286 Pin Configuration	3-75
3-73	Required Power, Ground and CAP Connections	3-76
3-74	Terminal Posts Provide Signal Access	3-77
3-75	I^2ICE™ Probe Cabling Requirements	3-77
4-1	Memory Interface Using Standard ALE Signal	4-7
4-2	Memory Interface Using Special Strobe Logic	4-8

Figure	Title	Page
4-3	Interleaved Memory Circuit	4-10
4-4	Address Strobe Generation	4-11
4-5	Memory Read Cycle Timing	4-13
4-6	Delaying Transceiver Enable	4-15
4-7	Memory Write Cycle Timing Using Standard ALE Strobe	4-16
4-8	Timing for Special Address Strobe Logic	4-19
4-9	Interleaved Memory Timing	4-21
4-10	ROM/PROM/EPROM Bus Interface	4-24
4-11	Generating Chip Selects for Devices without Output Enables	4-27
4-12	Generating Chip Selects for Devices with Output Enables	4-28
4-13	Static RAM Interface	4-29
4-14	80286-8207 Interfaces	4-31
4-15	8207 Single-Port Memory Subsystem	4-32
4-16	256K, 64K, and 16K RAM Address Connections	4-33
4-17	128K-Byte Memory Subsystem	4-34
4-18	External Shift Register Interface	4-35
4-19	128K-Byte Memory Subsystem Timing	4-36
4-20	8207 Memory Subsystem with ECC	4-38
5-1	Comparing Memory-Mapped and I/O-Mapped I/O	5-2
5-2	Restricted Address Regions	5-3
5-3	Memory-Mapped I/O Devices	5-4
5-4	Generating I/O Chip Selects	5-6
5-5	Bipolar PROM Decoder for 8-bit or 16-bit I/O Devices	5-7
5-6	16-Bit to 8-Bit Bus Conversion	5-7
5-7	16-Bit I/O Decode	5-8
5-8	Linear Selects for I/O Devices	5-9
5-9	I/O Cycle Model	5-10
5-10	I/O Read Cycle Timing	5-11
5-11	I/O Write Cycle Timing	5-11
5-12	Delaying Transceiver Enable	5-13
5-13	8274 MPSC Interface	5-16
5-14	8255A-5 Parallel Port Interface	5-18
5-15	Single 8259A Interrupt Controller Interface	5-20
5-16	Cascaded 8259A Interrupt Controller Interface	5-21
6-1	iAPX 286/20 System Configuration	6-2
6-2	Data Transfer Timing for the 80287	6-5
6-3	Data Channel Request and Acknowledge Timing	6-6
6-4	Software Routine to Recognize the 80287	6-8
7-1	Local Bus and MULTIBUS® Interface for the iAPX 286	7-4
7-2	Decoders Select the Local vs. the System Bus	7-5
7-3	Selecting the MULTIBUS® for Memory Operations	7-6
7-4	Memory-Mapping the MULTIBUS® I/O	7-7
7-5	Decoding Interrupt-Acknowledge Sequences	7-9

Figure	Title	Page
7-6	Byte-Swapping at the MULTIBUS® Interface	7-11
7-7	Implementing the Bus-Timeout Function	7-12
7-8	Two Bus Priority-Resolution Techniques	7-13
7-9	Bus Exchange Timing for the MULTIBUS®	7-14
7-10	Effects of Bus Contention on Bus Efficiency	7-16
7-11	Four Different 82289 Configurations	7-17
7-12	iAPX 286 Dual-Port Memory with MULTIBUS® Interface	7-21
7-13	Preventing Deadlock Between Dual-Port RAM and MULTIBUS®	7-22

Introduction 1

CHAPTER 1
INTRODUCTION

The iAPX 286 is a new high-performance, VLSI microprocessor system that supports multi-user reprogrammable and real-time multi-tasking applications. The iAPX 286 microsystem includes the 80286 microprocessor, the 80287 processor extension, and additional support components. The iAPX 286 architecture specifies how these components relate to each other, and is the key to the versatility of the iAPX 286 system.

iAPX 286 MICROSYSTEM COMPONENTS

The components that make up the iAPX 286 microsystem are designed to work together in modular combinations within the overall framework of the iAPX 286 architecture. System functions are distributed among specialized components, allowing designers to select an appropriate mix of components to fit the needs of their particular target system. This modular structure allows systems to grow in an orderly way to meet new needs, without adding unneeded capabilities or excessive cost.

Table 1-1 lists the components in the iAPX 286 microsystem and describes their functions.

Table 1-1. iAPX 286 Microsystem Components

Microprocessor	Description
80286 Central Processing Unit (CPU)	16-bit high performance microprocessor with on-chip memory management and memory protection
80287 Numeric Processor Extension (NPX)	High-performance numeric extension to the 80286 CPU
Support Component	**Description**
8259A Programmable Interrupt Controller	Provides interrupt control and priority management for the CPU
8282 Octal Latch (Non-inverting)	Latches address and increases drive on address bus
8283 Octal Latch (Inverting)	Latches address for inverted-sense buses like the IEEE 796 Multibus
82284 Clock Generator and Driver	Generates system timing functions, including system clock, RESET, and Ready synchronization
8286 Octal Bus Transceiver	Increases drive on data bus
8287 Octal Bus Transceiver (Inverting)	Buffers data bus for inverted-sense buses like the IEEE 796 Multibus
82288 Bus Controller	Generates bus command signals
82289 Bus Arbiter	Controls access of microprocessors to multi-master bus

Microprocessors

At the center of the iAPX 286 microsystem is the 80286 Central Processing Unit (CPU). The 80286 is a high-performance microprocessor with a 16-bit external data path and up to 16 megabytes of directly-addressable physical memory; up to one gigabyte of virtual memory space is available to each user. The standard operating speed of the 80286 is 8 MHz; a 6-MHz version of the 80286 is also available.

The 80286 operates in two modes: Real-Address Mode and Protected Virtual-Address Mode.

- In Real-Address Mode, the 80286 is fully code-compatible with the iAPX 86 and 88 microprocessors. All 8086 and 8088 instructions execute on the 80286 at a much faster rate. An 8 MHz 80286 executes instructions up to six times faster than a 5-MHz 8086.
- In Protected Virtual-Address Mode, the 80286 is also compatible with the 8086 instruction set. Protected Virtual-Address Mode allows use of the 80286's built-in memory protection and management capabilities and virtual-memory support.

The 80287 Numeric Processor Extension is an iAPX 286 processor extension that uses the 80286 to fetch instructions and transfer operands. The 80287 extends the numeric processing abilities of the 80286 to include 8-, 16-, 32-, 64-, and 80-bit integer and floating-point data types and to perform common transcendental functions. The 80287 is compatible with the proposed IEEE 754 Floating-Point Standard.

Interrupt Controller

Interrupt management for an iAPX 286 system is provided by the 8259A Programmable Interrupt Controller. Interrupts from up to eight sources are accepted by the 8259A; up to 64 requests can be accommodated by cascading several 8259A devices. The 8259A typically resolves priority between active interrupts, interrupts the CPU, and passes a code to the CPU to identify the interrupting source. Programmable features of this device allow it to be used in a variety of ways to fit the interrupt requirements of the particular system.

Bus Interface Components

Bus interface components connect the iAPX 286 processors to memory and peripheral devices. The use of these components in a system is based on the requirements of the particular system configuration.

The 82284 Clock Generator is a bipolar device that generates timing for the iAPX 286 processors and support components. The device supplies the internal clock to the processors and also provides a TTL-level half-frequency peripheral clock signal to other devices in the system. The Clock Generator also synchronizes RESET and Ready signals for the processors and support devices.

Addresses and control signals are latched and held by 8282 or 8283 Octal Latches. These latches are bipolar devices that satisfy the high AC and DC drive requirements of larger systems.

The 8286 and 8287 Octal Bus Transceivers are bipolar, high-performance bi-directional data buffers that satisfy the high AC and DC drive requirements of larger systems.

The 82288 Bus Controller is an HMOS device that decodes status lines from the 80286 CPU to generate bus command signals. These command signals identify and control the bus cycles that are performed. The 82288 also provides address latch and data buffer control signals for the 8282/8283 Address

Latches and the 8286/8287 Bus Transceivers. The Bus Controller's command outputs provide the high AC and DC drive required for large systems, while control inputs allow tailoring of the command timing to accommodate a wide variety of timing requirements.

Access by the iAPX 286 to a multi-master system bus is controlled by the HMOS 82289 Bus Arbiter. A multi-master system bus connects memory and peripheral resources that are shared by two or more processing elements. Bus Arbiters for each processor can use one of several priority-resolving techniques to ensure that only one processor is driving the bus at any given time. The 82289 Bus Arbiter supports bus arbitration signals that are fully-compatible with the IEEE 796 Multibus standard.

iAPX 286 System Architecture 2

CHAPTER 2
iAPX 286 SYSTEM ARCHITECTURE

The iAPX 286 microsystem supports a very flexible architecture that opens up a wide range of possibilities for system designers. This chapter describes the iAPX 286 system architecture and examines some of the possible configurations this architecture supports.

- The first section of this chapter describes the organization of the iAPX 286 bus. This parallel bus structure forms the basis for many of the configuration possibilities of the iAPX 286 architecture.
- The second section introduces memory subsystems for the iAPX 286. Memory is a crucial part of every microprocessing system; it is used for the storage of program instructions and for the holding of processing data and information.
- The third section discusses the possible I/O configurations of an iAPX 286 system. The iAPX 286 can directly address a large number of peripheral devices, or alternatively, I/O processing tasks can be off-loaded to dedicated processing subsystems.
- The fourth section introduces the 80287 Numeric Processor Extension and briefly describes how the 80287 operates in conjunction with the 80286 CPU.
- The final section of this chapter describes the architectural features of the iAPX 286 microsystem which support multi-processing. Multi-processing provides virtually limitless possibilities for increasing the performance of iAPX 286 systems through the modular expansion of processing capabilities.

iAPX 286 BUS ORGANIZATION

The iAPX 286 system architecture is centered around a parallel bus structure that connects the iAPX 286 processor to memory and I/O resources. Across this bus, the iAPX 286 processor fetches program instructions, manipulates stored information, and interacts with external I/O devices. Figure 2-1 shows this basic iAPX 286 bus architecture.

The iAPX 286 architecture contains two types of buses: the local bus and the system bus.

- The local bus connects the iAPX 286 processor with processor extensions and other processing elements. Private memory and I/O devices connect to a buffered version of this local bus, and are available only to processing elements residing on the local bus.
- The system bus connects the iAPX 286 processor to public memory and I/O resources. These public resources are typically shared among the iAPX 286 processor and other bus masters that also connect to the system bus.

Specialized bus interface components connect the iAPX 286 local bus to both the buffered local bus and the system bus. These interface components process the signals that come from the iAPX 286 local bus and generate appropriate signals for use on the buffered local bus or the public system bus.

The iAPX 286 Local Bus

The iAPX 286 local bus is formed from the signals generated by the 80286 CPU. These signals include address, data, and control or status information that directs the operation of the local bus. All iAPX 286 systems have at least one local bus containing the 80286 CPU.

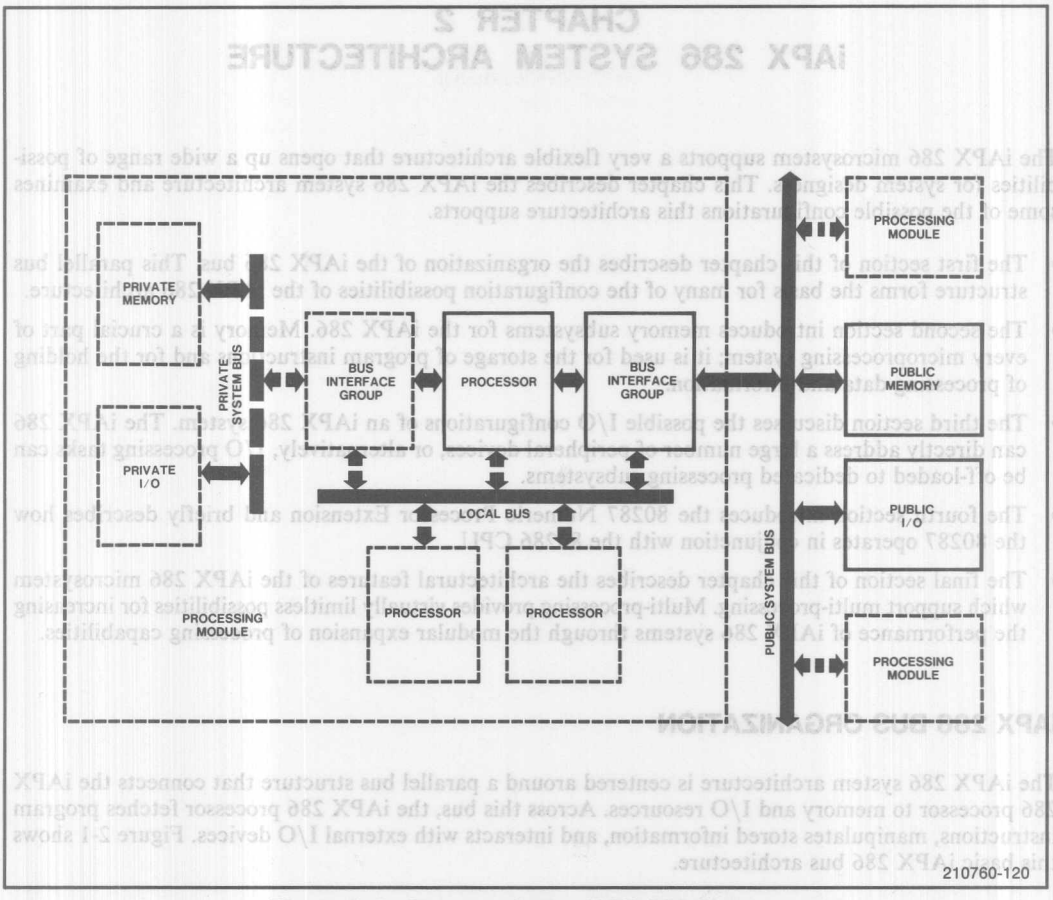


Figure 2-1. Representative iAPX 286 System

The 80286 provides on-chip arbitration logic and external control signals that allow it to share the local bus with other processing elements. Examples of other processing elements that connect to the local bus are processor extensions, Direct Memory Access (DMA) devices, or other specialized processors. These arbitration functions provide a cost-effective way to structure a small, multi-processing system, since the same bus interface components are shared by all of the processing elements on the local bus. Additional arbitration logic is not required for additional processors on the local bus.

The Buffered Local Bus

A buffered local bus is just that: a buffered version of the iAPX 286 local bus which contains private memory and I/O resources that are accessible only to processors on that local bus. All but the smallest of iAPX 286 systems contain a buffered version of the local bus. Since the buffered local bus takes full advantage of the iAPX 286 local bus timing, and since the iAPX 286 processor does not have to contend with other system processing elements for the use of these resources, the maximum system performance is typically obtained when the iAPX 286 is executing programs from memory that resides on this buffered local bus.

The System Bus

The system bus, like the buffered local bus, is composed of buffered versions of the local bus signals. Unlike a buffered local bus, however, the public system bus connects memory and I/O resources that are shared among processors belonging to more than one local bus.

By providing easy shared access to processing resources on the system bus, the public system bus allows multi-processor systems to communicate among themselves and to avoid a needless duplication of resources.

Bus Interface Groups

Specialized bus interface components are used to translate the iAPX 286 local bus signals onto the buffered bus to connect the iAPX 286 local bus to either a buffered local bus or to a public system bus.

These bus interface components include the 82288 Bus Controller, 8282 and 8283 Octal Address Latches, 8286 and 8287 Octal Bus Transceivers, and the 82289 Bus Arbiter. These components can be mixed and matched as required to connect the 80286 to buffered local or system buses. In fact, the modular nature of the iAPX 286 bus structure allows as many local and system buses as required by the target system.

Figure 2-2 shows an example of an 80286 connected to a multi-master (more than one processor) public system bus. Address latches and data transceivers transfer the local bus address and data onto the system bus, respectively. The 82288 Bus Controller translates the 80286 local bus signals to provide the system bus command signals. The 82289 Bus Arbiter provides the arbitration and control functions necessary to ensure that only one master has control of the system bus at any given time.

Figure 2-3 shows an iAPX 286 connected to both a multi-master system bus and to a buffered local bus. Separate Bus Controllers, address latches, and data transceivers are used to implement each of the two bus interfaces. No Bus Arbiter is required for the buffered local bus interface, because the 80286 has exclusive control of the buffered local bus.

In a system having more than one bus, address decoders typically select which of the available buses is being requested when the iAPX 286 performs a bus operation. The appropriate Bus Controller and Bus Arbiter, if any, are activated and the iAPX 286 bus operation continues to completion.

Chapter Three describes the operation of the iAPX 286 local bus in more detail. Chapter Seven discusses the considerations for designing interfaces between the iAPX 286 and a public system bus.

The following sections provide a closer look at some of the configuration possibilities for iAPX 286 systems.

MEMORY SUBSYSTEMS FOR THE iAPX 286

An iAPX 286 processor operating in Protected Virtual-Address mode can directly address as much as 16 megabytes of physical memory. This large physical address space can be segmented into a number of separate memory subsystems, with each having different characteristics and functions, but all memory subsystems being directly accessible by the iAPX 286 processor.

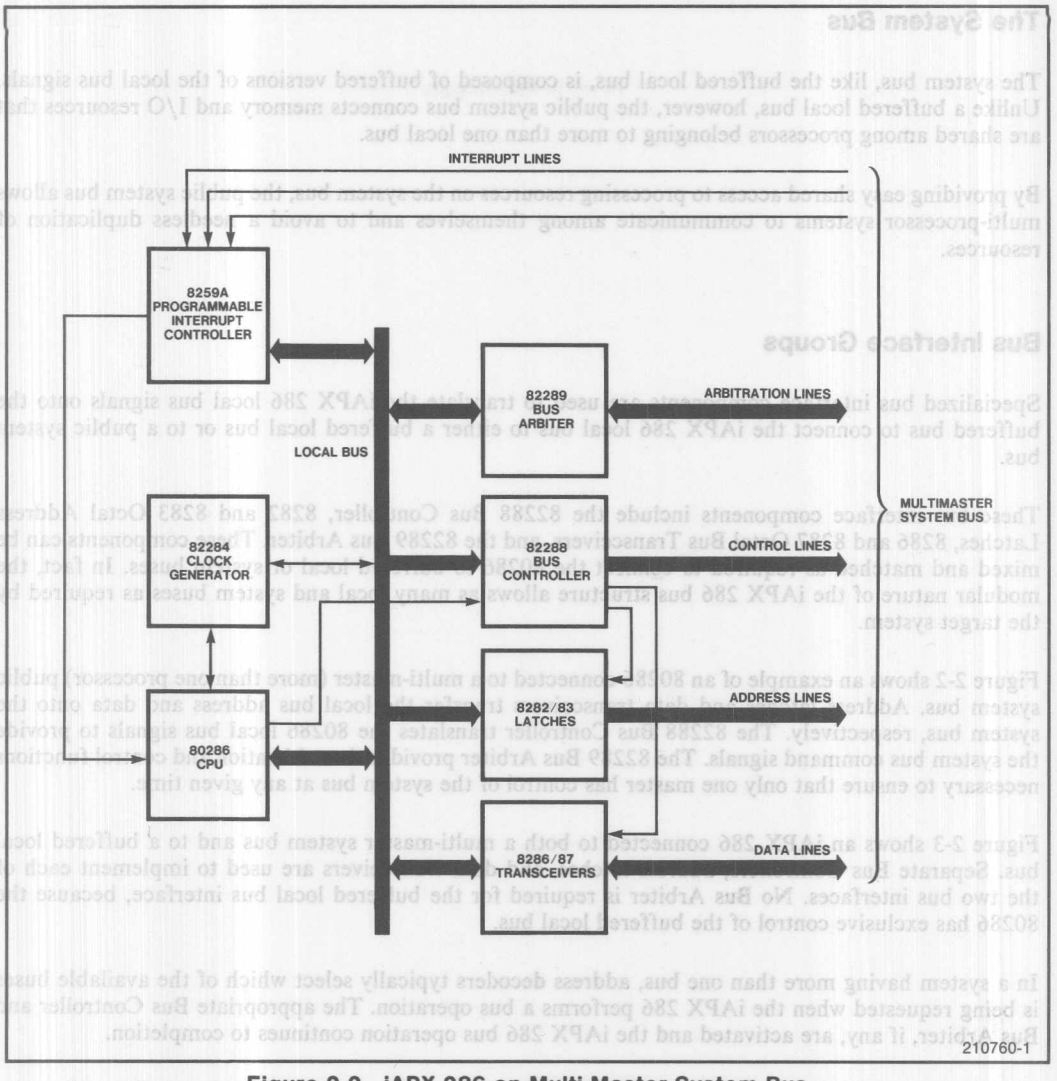


Figure 2-2. iAPX 286 on Multi-Master System Bus

Individual memory systems for the iAPX 286 can be configured either as local memory, accessible by an 80286 and other processors on a single local bus, or as system memory, to be shared by multiple independent processors each having their own local bus:

- Local memory resides on a buffered local bus and is available to processors on that local bus only.
- System memory resides on a public system bus and is available to all processors that interface to the public bus.
- A third alternative that combines the advantages of both local and system memory is to configure the same memory to be accessible both as local memory from a local bus, and as system memory from a public system bus. Such a memory system is called a dual-port memory.

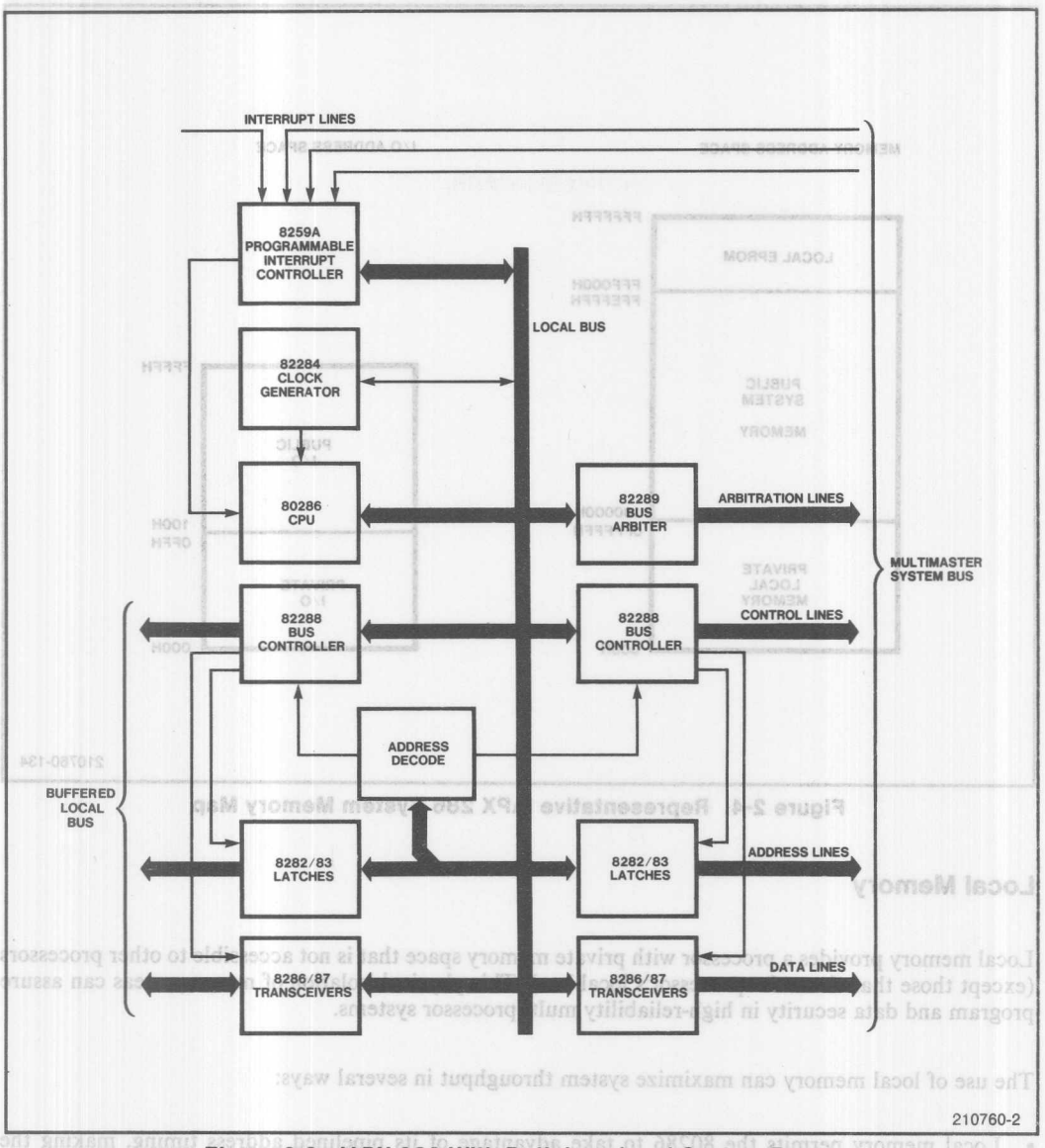


Figure 2-3. iAPX 286 With Both Local and System Buses

The iAPX 286 architecture allows any of these configurations to be used in a single system; the use of one or all memory approaches depends on the needs of the target system.

Typical iAPX 286 systems will use some combination of both local and system memory (the system in Figure 1-1 shows both). In systems configured with memory on both the buffered local bus and the system bus, the selection of which bus is accessed during a given memory operation is based on the memory or I/O address. Figure 2-4 shows an example memory and I/O address map for the iAPX 286 system shown in Figure 2-1.

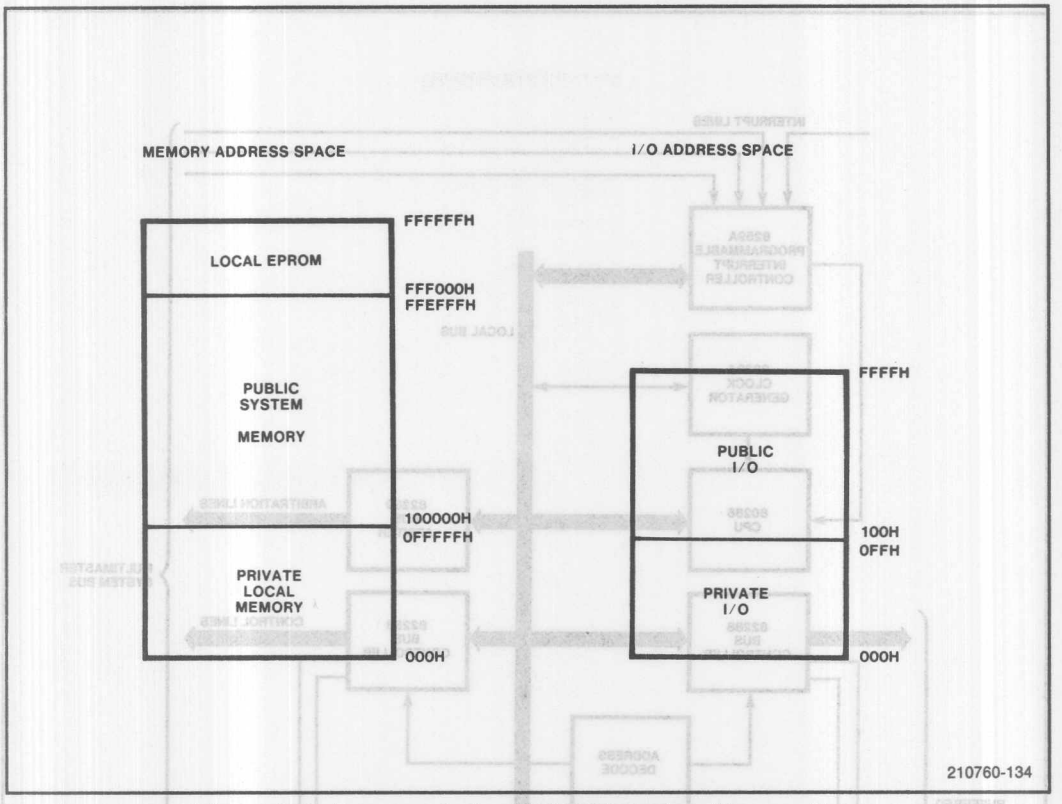


Figure 2-4. Representative iAPX 286 System Memory Map

Local Memory

Local memory provides a processor with private memory space that is not accessible to other processors (except those that share the processor's local bus). This physical isolation of memory areas can assure program and data security in high-reliability multi-processor systems.

The use of local memory can maximize system throughput in several ways:

- Local memory permits the 80286 to take advantage of its pipelined address timing, making the most efficient use of the bus and performing memory operations in the least possible time.
- Since the local memory is not shared with other processors, no access delays are incurred, whereas access delays usually happen when several processors contend for use of the same resources.
- In systems where several processors each have their own private memory space, multiple tasks can execute in parallel because each processor is fetching instructions on a separate data path.

The tradeoffs associated with using local memory in multi-processor systems include the cost of implementing a separate memory subsystem for each processor.

System Memory

System memory provides a shared memory space that can be accessed by all processors connected to a public system bus. This shared memory allows multiple processors to communicate with one another and efficiently pass blocks of data between their separate tasks.

Shared system memory is often a cost-effective alternative to using local memory when individual processors may occasionally require a relatively large physical memory space. Since the same memory is shared between multiple processors, the total system cost can be considerably less than if local memory for each processor were sized to accommodate their largest respective memory requirements.

Since system memory typically can be expanded simply by installing additional memory boards, this modular flexibility makes system memory attractive for many applications.

The tradeoffs associated with using shared system memory instead of local memory in multi-processor systems include the fact that access to system memory may be slower than access to local memory because processors must contend with each other for access to the system bus.

When several processors require use of the same memory resources at the same time, memory access time (and therefore system throughput) can be significantly reduced.

The risk of data corruption with system memory is also greater in systems that use other processors along with an iAPX 286 because one processor can possibly overwrite data being used by another processor. Software protocols usually can avoid this problem, however.

Dual-Port Memory

Dual-port memory is a single memory subsystem that combines many of the advantages of both local and system memory. Dual-port memory appears both as local memory to processors on a single local bus, and as system memory to other processors in the system. Figure 2-5 shows an iAPX 286 system that uses dual-port memory.

Dual-port memory permits the local processor to have high-speed access to the dual-port memory without tying up the public system bus. In this way, iAPX 286 systems can access the dual-port memory as local cache memory without affecting other processors using other system resources on the system bus.

The dual-port memory is also accessible by other processors using the system bus, providing a shared memory space that can be used for inter-processor communications or other functions.

If necessary, address-mapping circuitry can make the memory appear in different address ranges for local and system bus processors. Address-mapping can also be used to permit other system processors access to only a portion of the dual-port memory, reserving portions of the memory array for exclusive use by the local processor.

Offsetting these advantages is the fact that dual-port memory typically is more complex than a (single-port) local or system memory subsystem. Dual-port memories require arbitration logic to ensure that only one of the two buses serving the memory can gain access at one time. Figure 2-6 gives a visualization of this arbitration requirement. Fortunately, the Intel 8207 Advanced Dynamic RAM Controller (ADRC) greatly simplifies design of dual-port memory subsystems.

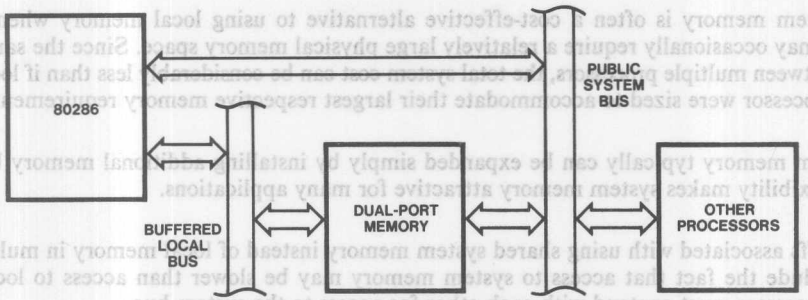


Figure 2-5. Dual-Port Memory

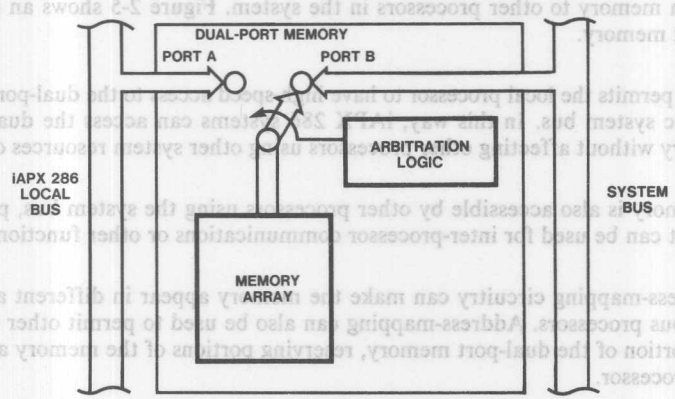


Figure 2-6. Arbitration Logic for a Dual-Port Memory

The 8207 Advanced Dynamic RAM Controller is a high-performance dynamic RAM controller designed to easily interface 16K, 64K, and 265K dynamic RAM devices to microprocessor systems. On-chip arbitration and synchronization logic implements the dual-port function, allowing two different buses to independently access the RAM array. The 8207 DRAM controller has a synchronous mode of operation that specifically supports the 80286 processor.

By combining RAM control and dual-port arbitration functions on a single chip, the 8207 ADRC provides an easy way to implement dual-port memory in an iAPX 286 system. Figure 2-7 shows an 8207-based dual-port memory subsystem connected to an iAPX 286 subsystem and a Multibus system bus interface.

The advantages of dual-port memories, described above, make dual-port memory a more effective choice for multi-processor applications where using strictly local or system memory would significantly

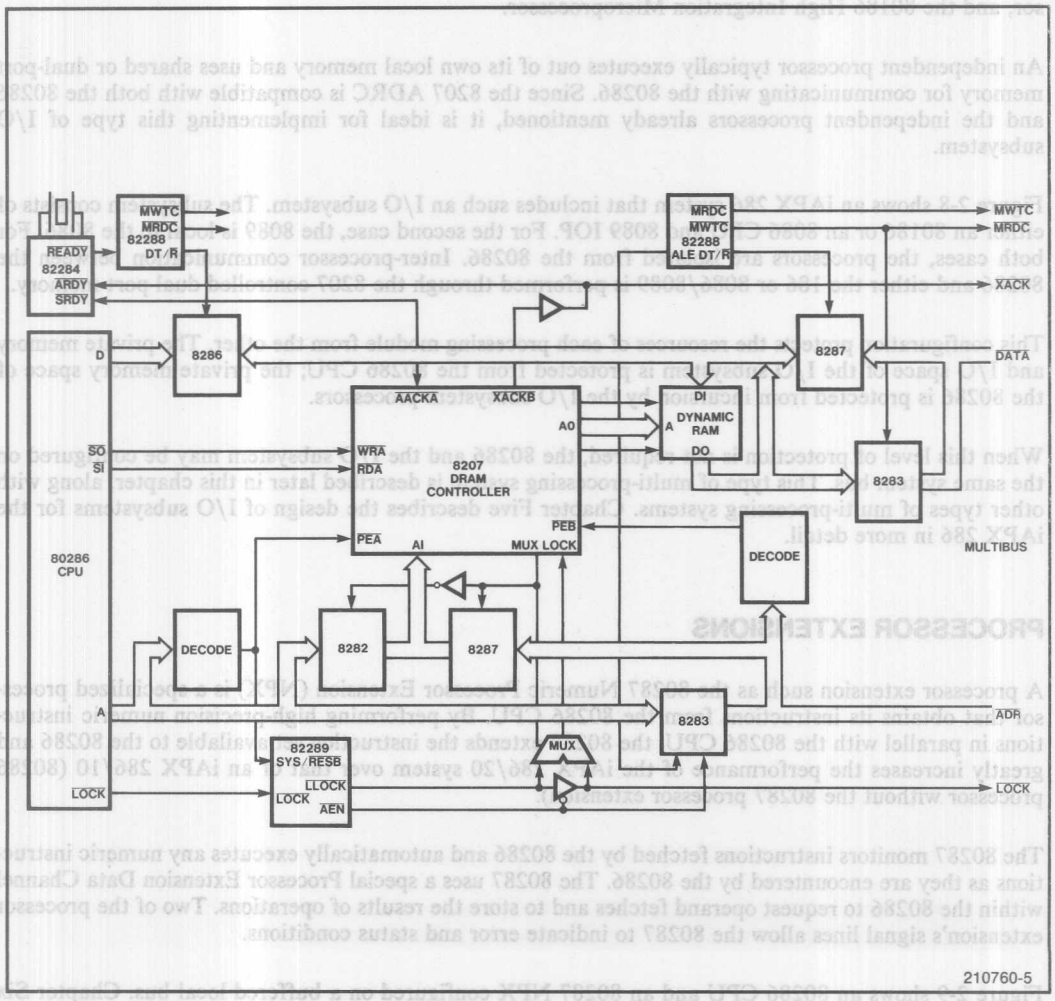


Figure 2-7. 8207 Dual-Port Subsystem

impact system cost and/or performance. Chapter Four describes the design of memory subsystems for he iAPX 286 in more detail.

I/O SUBSYSTEMS FOR THE iAPX 286

The iAPX 286 processor can directly address as many as 32,768 16-bit I/O devices, or 65,536 8-bit I/O devices, or combinations of the two. This large I/O address space is completely separate from the memory address space, as shown previously in Figure 2-4.

In addition to accessing individual peripheral devices, an iAPX 286 system that requires extensive I/O capability can offload its I/O processing tasks to one or more dedicated, independent processors. An independent processor is one that executes an instruction stream separate from the 80286 CPU. Examples of independent processors include the 8086 and 8088 CPU's, the 8089 Input/Output Processor, and the 80186 High Integration Microprocessor.

An independent processor typically executes out of its own local memory and uses shared or dual-port memory for communicating with the 80286. Since the 8207 ADRC is compatible with both the 80286 and the independent processors already mentioned, it is ideal for implementing this type of I/O subsystem.

Figure 2-8 shows an iAPX 286 system that includes such an I/O subsystem. The subsystem consists of either an 80186 or an 8086 CPU and 8089 IOP. For the second case, the 8089 is local to the 8086. For both cases, the processors are isolated from the 80286. Inter-processor communication between the 80286 and either the 186 or 8086/8089 is performed through the 8207-controlled dual-port memory.

This configuration protects the resources of each processing module from the other. The private memory and I/O space of the I/O subsystem is protected from the 80286 CPU; the private memory space of the 80286 is protected from incursion by the I/O subsystem processors.

When this level of protection is not required, the 80286 and the I/O subsystem may be configured on the same system bus. This type of multi-processing system is described later in this chapter, along with other types of multi-processing systems. Chapter Five describes the design of I/O subsystems for the iAPX 286 in more detail.

PROCESSOR EXTENSIONS

A processor extension such as the 80287 Numeric Processor Extension (NPX) is a specialized processor that obtains its instructions from the 80286 CPU. By performing high-precision numeric instructions in parallel with the 80286 CPU, the 80287 extends the instruction set available to the 80286 and greatly increases the performance of the iAPX 286/20 system over that of an iAPX 286/10 (80286 processor without the 80287 processor extension).

The 80287 monitors instructions fetched by the 80286 and automatically executes any numeric instructions as they are encountered by the 80286. The 80287 uses a special Processor Extension Data Channel within the 80286 to request operand fetches and to store the results of operations. Two of the processor extension's signal lines allow the 80287 to indicate error and status conditions.

Figure 2-9 shows an 80286 CPU and an 80287 NPX configured on a buffered local bus. Chapter Six describes in detail the design of iAPX 286 systems using the 80287 Numeric Processor Extension.

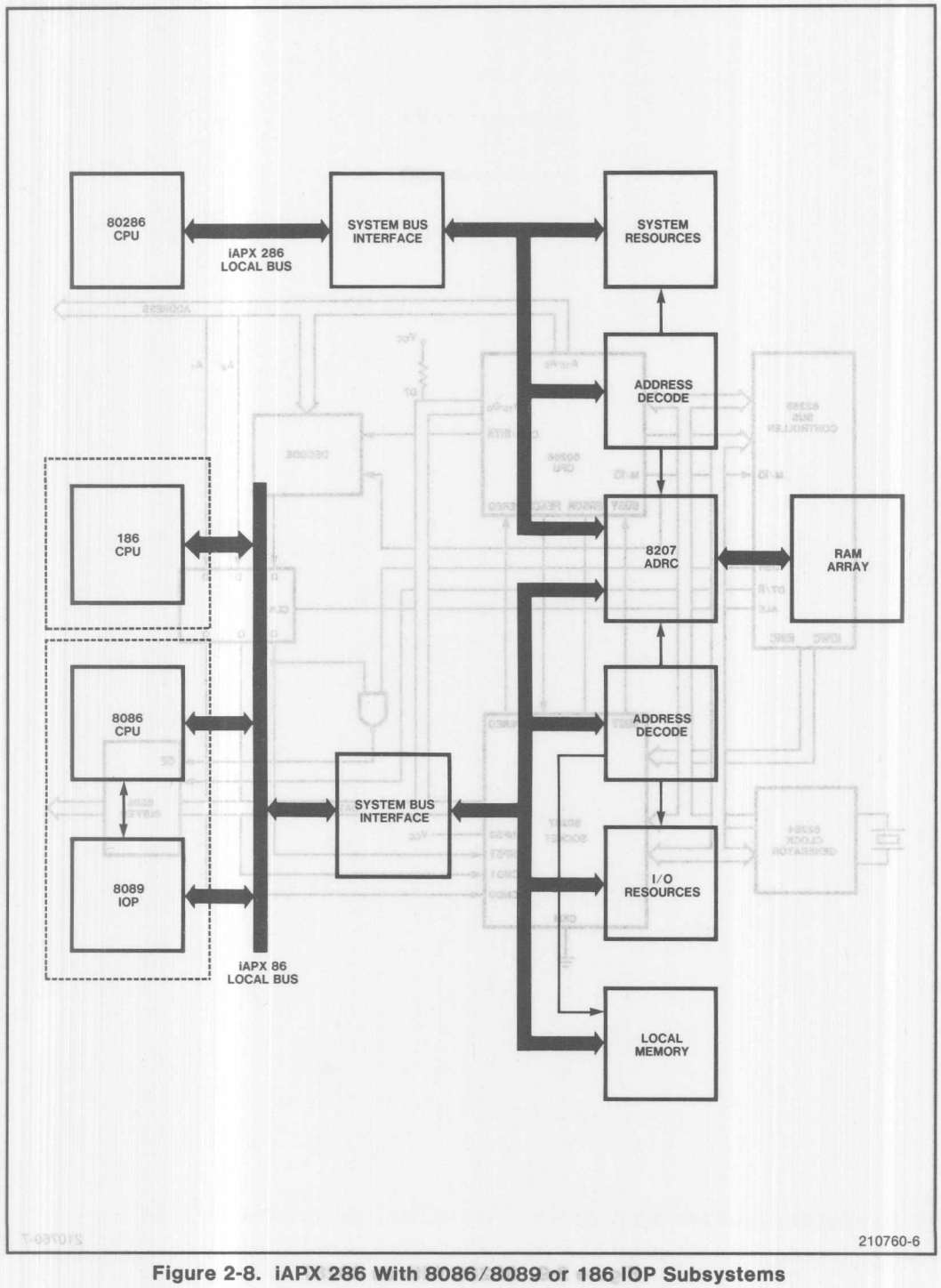


Figure 2-8. iAPX 286 With 8086/8089 or 186 IOP Subsystems

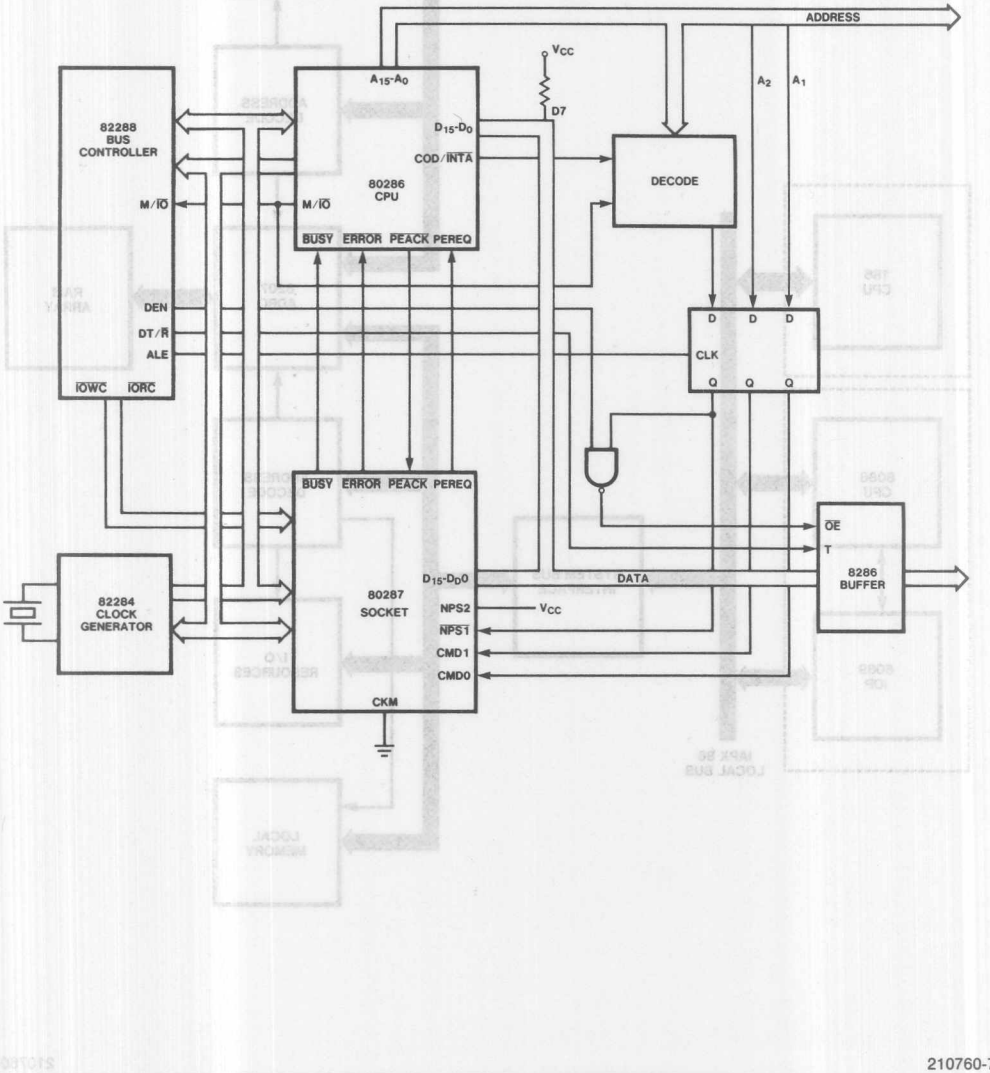


Figure 2-9. 80286 With an 80287

A single 80286 can provide a performance increase of up to 6 times that of a single 8086 CPU. In large systems requiring even greater performance, system throughput can be increased even further by employing multiple processors. By distributing system functions among multiple processors, significant advantages can be gained:

- System tasks can be allocated to special-purpose processors whose designs are optimized to perform those tasks simply and efficiently.
- Very high levels of performance can be attained when multiple processors execute simultaneously (parallel processing).
- System reliability can be improved by isolating system functions so that a failure or error which occurs in one part of the system has a limited effect on the rest of the system.
- Multi-processing promotes partitioning the system into modules, breaking system development into more manageable tasks and permitting the parallel development of subsystems. Partitioning also helps to isolate the effects of system modifications to a specific module.

Designers of multi-processing systems typically have been faced with the two classic problems whenever more than one processor shares a common bus: bus arbitration and mutual exclusion. The iAPX 286 system architecture provides built-in solutions that virtually eliminate these problems.

Bus Arbitration

Bus arbitration is the means whereby one processing element gains control of a shared common bus from another processing element. In an iAPX 286 system, this shared bus may be either the iAPX 286 local bus (if more than one processor is configured on the local bus), or a public system bus that is shared between multiple processing elements. Bus arbitration for the control of the local bus is performed directly by the 80286 CPU. For the control of a public system bus, specialized support devices are used to provide the arbitration function.

At the local bus level, the 80286 provides bus arbitration through its on-chip arbitration logic and control signals. Two control signals, called HOLD and HLDA (hold and hold acknowledge), are provided, and are typically used with dedicated bus masters such as a Direct Memory Access (DMA) Controller.

Figure 2-10 shows an 80286 configured with a DMA controller on the local bus. The DMA controller requests control of the local bus by asserting the HOLD signal to the 80286. The CPU responds by relinquishing control of the bus and sending the controller an acknowledge signal (HLDA). The 82257 Advanced DMA Controller, a high-performance DMA controller specifically designed to support the 80286 local bus protocols, will be available in 1984.

At the system bus level, bus arbitration for iAPX 286 systems typically is performed by the 82289 Bus Arbiter. The Bus Arbiter connects to the 80286 CPU and controls iAPX 286 access to a multi-master system bus such as the IEEE 796 Multibus. The 82289 supports a number of techniques for resolving requests from multiple masters on the system bus, including both serial and parallel priority resolution. Each iAPX 286 subsystem connected to the public system bus has its own Bus Arbiter.

For other (non-80286) processors connected to the public system bus, the 8289 Bus Arbiter provides the necessary bus arbitration functions. The 8289 Bus Arbiter supports the 8086 and 8088 CPU's, the 80186 High Integration Microprocessor, and the 8089 Input/Output Processor. The 8289 Bus Arbiter is compatible with the 82289 bus control signals and the Multibus interface.

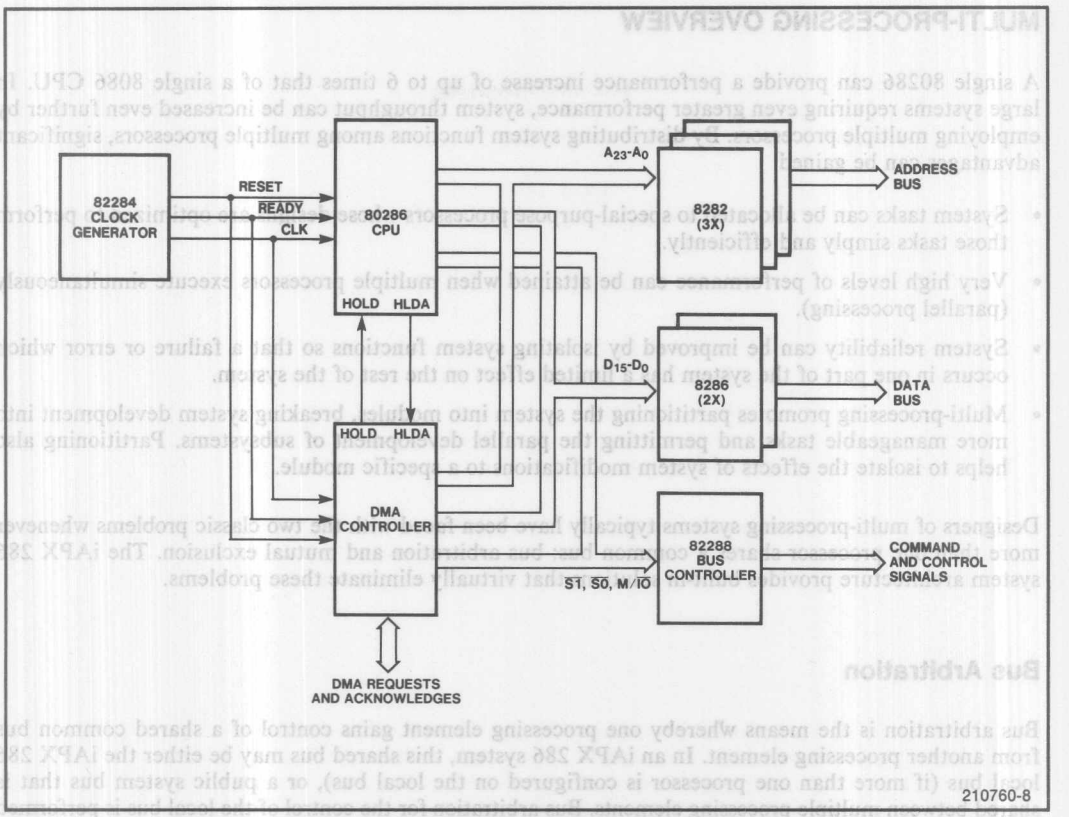


Figure 2-10. 80286 with DMA Controller

Figure 2-11 illustrates the use of the 82289 Bus Arbiter in implementing a system bus interface. iAPX 286 designs using the 82289 Bus Arbiter are described in greater detail in Chapter Seven.

Mutual Exclusion

Mutual exclusion is a property of a shared resource which assures that only one processing element uses the resource at one time. Typically, mutual exclusion means that one processor has the ability to prevent other processing elements from accessing a shared resource (such as the bus) until the processor has finished using the resource.

In many cases, mutual exclusion can be accomplished in software through the use of semaphores. In other cases, however, such as when several memory operations must be completed as a unit, support for mutual exclusion must be provided in hardware. The iAPX 286 microsystem provides basic hardware support for mutual exclusion through the 80286 $\overline{\text{LOCK}}$ signal.

The 80286 $\overline{\text{LOCK}}$ signal is activated automatically during specific CPU operations, or explicitly through use of the ASM-286 LOCK prefix instruction, to prevent other processors from accessing a shared resource. The CPU operations which assert an active LOCK signal include interrupt-acknowledge sequences, the ASM-286 XCHG instruction, and descriptor-table accesses.

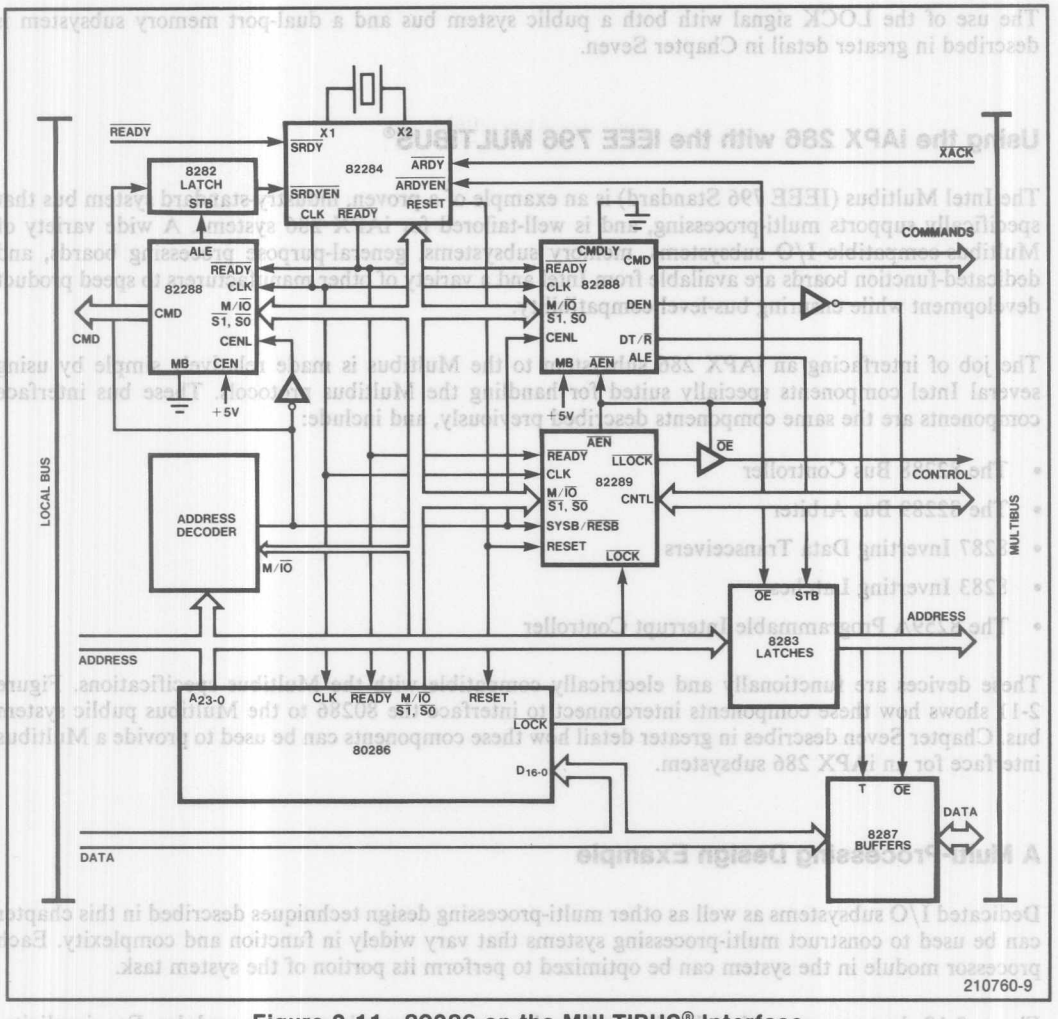


Figure 2-11. 80286 on the MULTIBUS® Interface

The 82289 Bus Arbiter and 8207 Advanced Dynamic RAM Controller both have inputs that connect to the 80286 $\overline{\text{LOCK}}$ signal. When its Lock input is active, the Bus Arbiter will not relinquish control of the system bus, preventing other system processors from using the system bus. In a dual-port memory subsystem, the 8207 ADRC $\overline{\text{LOCK}}$ input restricts memory access to a single processor and so prevents other processors from altering memory. Public system buses such as the IEEE 796 Multibus define a similar LOCK signal, which is used to restrict access to the system bus resources.

The ability to exclude other processors from a shared resource in selected situations can provide a high level of performance and integrity in an iAPX 286 system. For example, context switching and descriptor-table loading can be performed quickly and safely by $\overline{\text{LOCK}}$ing the transfer sequence because delays due to bus contention on the system bus are minimized and all data is protected from other processors until the sequence is complete. Interrupt responses, semaphore accesses, and $\overline{\text{LOCK}}$ed data transfers are also fast and reliable in an iAPX 286 system.

The use of the $\overline{\text{LOCK}}$ signal with both a public system bus and a dual-port memory subsystem is described in greater detail in Chapter Seven.

Using the iAPX 286 with the IEEE 796 MULTIBUS®

The Intel Multibus (IEEE 796 Standard) is an example of a proven, industry-standard system bus that specifically supports multi-processing, and is well-tailored for iAPX 286 systems. A wide variety of Multibus-compatible I/O subsystems, memory subsystems, general-purpose processing boards, and dedicated-function boards are available from Intel and a variety of other manufacturers to speed product development while ensuring bus-level compatibility.

The job of interfacing an iAPX 286 subsystem to the Multibus is made relatively simple by using several Intel components specially suited for handling the Multibus protocols. These bus interface components are the same components described previously, and include:

- The 82288 Bus Controller
- The 82289 Bus Arbiter
- 8287 Inverting Data Transceivers
- 8283 Inverting Latches
- The 8259A Programmable Interrupt Controller

These devices are functionally and electrically compatible with the Multibus specifications. Figure 2-11 shows how these components interconnect to interface the 80286 to the Multibus public system bus. Chapter Seven describes in greater detail how these components can be used to provide a Multibus interface for an iAPX 286 subsystem.

A Multi-Processing Design Example

Dedicated I/O subsystems as well as other multi-processing design techniques described in this chapter can be used to construct multi-processing systems that vary widely in function and complexity. Each processor module in the system can be optimized to perform its portion of the system task.

Figure 2-12 shows a system that distributes functions among several processor modules. For simplicity, bus interface components are not shown.

The supervisor module consists of a single 80286 CPU and local memory. The supervisor controls the system, primarily responding to interrupts and dispatching tasks to the other modules. The supervisor executes code out of local memory that is inaccessible to the other processors in the system. System memory, accessible to all of the processors connected to the public system bus, is used for messages and common buffers.

Each graphics module supports a graphics CRT terminal and contains an 80286 CPU and 80287 NPX (iAPX 286/20 configuration), local memory, and local I/O.

The database module is responsible for maintaining all system files; it consists of an 80286 CPU with an iAPX 86/11 (8086 CPU and 8089 IOP) subsystem that controls the actual storage devices. Both the iAPX 286/10 processor and the iAPX 86/11 subsystem in the database module have their own local memory. A dual-port memory connects the iAPX 286/10 to the subsystem processors.

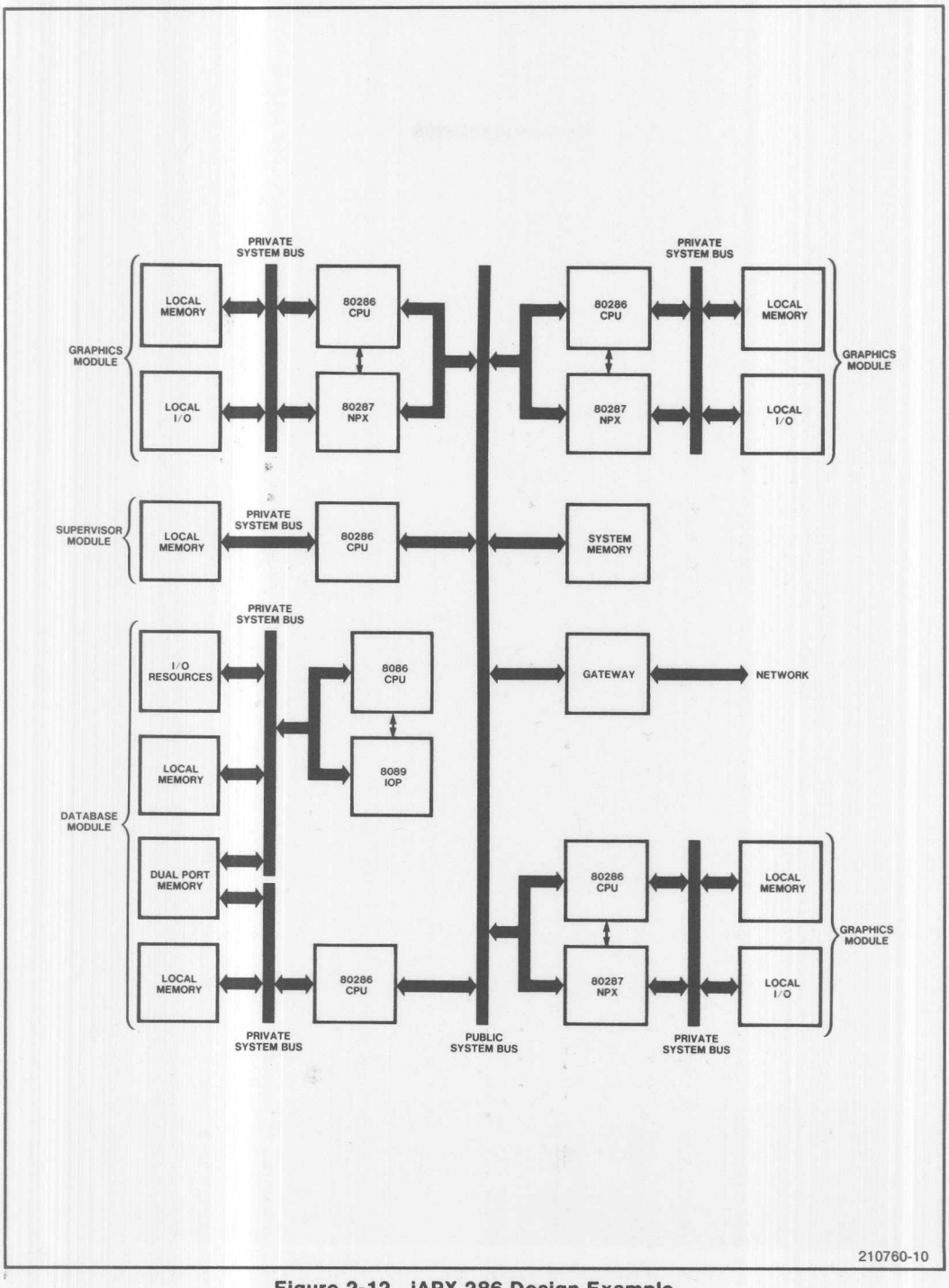


Figure 2-12. iAPX 286 Design Example

Figure 2-12. iAPX 286 Design Example

The iAPX 286 Local Bus 3

CHAPTER 3
THE iAPX 286 LOCAL BUS

The iAPX 286 local bus connects the 80286 processor to memory and peripheral devices, and forms the backbone of any iAPX 286 system. This chapter introduces the 80286 processor and describes the operation of the iAPX 286 local bus:

- The first section of this chapter introduces the 80286 processor. The four internal processing units that make up the 80286 processor are described, as are the two operating modes of the 80286.
- The second section introduces the iAPX 286 local bus, and gives an overview of the principles used by the 80286 in managing resources on the local bus:
 - — The organization of the bus is described first, covering memory addressing, I/O, and interrupts.
 - — The timing of the local bus is described, starting with a description of the local bus states, describing the 80286 bus operations that are composed of two or more bus states, and finally describing the way in which the 80286 uses the various bus operations.
 - — Finally, specific bus interface components are introduced which greatly ease the design of iAPX 286 local bus systems. Two of these interface components include the 82284 Clock Generator and the 82288 Bus Controller.
- The third section contains specific design information on generating timing for an iAPX 286 local bus, using the 82284 Clock Generator. The generation of the three principle timing signals (the system CLK, RESET, and $\overline{\text{READY}}$) is described in detail.
- The fourth section describes how to generate iAPX 286 local bus control signals using the 82288 Bus Controller.
- The fifth section describes additional design alternatives concerning the connection of memory, I/O, and other devices to the local bus. These design considerations include:
 - — Address decoding and data buffering.
 - — Connecting other bus masters to the local bus.
 - — Initializing the 80286.
 - — Detailed local bus timing.
 - — Physical design considerations including layout, packaging, power and ground connections.
- Finally, the last section of this chapter introduces the iLBX bus, a high-performance local bus standard to allow the modular expansion of the iAPX 286 local bus onto multiple boards.

INTRODUCTION TO THE 80286 CENTRAL PROCESSING UNIT

The 80286 Central Processing Unit (CPU) is an advanced, high-performance 16-bit microprocessor that is optimized for use in multiple-user and multi-tasking systems. The 80286 has built-in memory management and protection capabilities which permit operating system and task isolation as well as program and data isolation within tasks. In addition, a highly-efficient pipeline architecture in both the CPU itself and in the local bus protocols serves to maximize iAPX 286 system throughput and performance, while minimizing the impact on bus- and memory-speed requirements.

Processor Architecture

As shown in Figure 3-1, the pipelined 80286 architecture consists of four independent processing units; these four units operate in parallel to maximize CPU performance.

THE BUS UNIT

The Bus Unit (BU) performs all bus operations for the CPU, generating the address, data, and command signals required to access external memory and I/O. The Bus Unit also controls the interface to processor extensions and other local bus masters. Most of the local bus signals interface directly to the BU.

When not performing other bus duties, the Bus Unit "looks ahead" and pre-fetches instructions from memory. When prefetching, the Bus Unit assumes that program execution proceeds sequentially; that is, the next instruction follows the preceding one in memory. When the prefetcher reaches the limit of the code segment, it stops prefetching instructions. If a program transfer causes execution to continue from a new program location, the Bus Unit resets the queue and immediately begins fetching instructions from the new program location.

The Bus Unit stores these instructions in a 6-byte prefetch queue to be used later by the Instruction Unit. By prefetching instructions, the BU eliminates the idle time that can occur when the CPU must wait for the next sequential instruction to be fetched from memory.

THE INSTRUCTION UNIT

The Instruction Unit (IU) receives instructions from the prefetch queue, decodes them, and places these fully-decoded instructions into a 3-deep instruction queue for use by the Execution Unit.

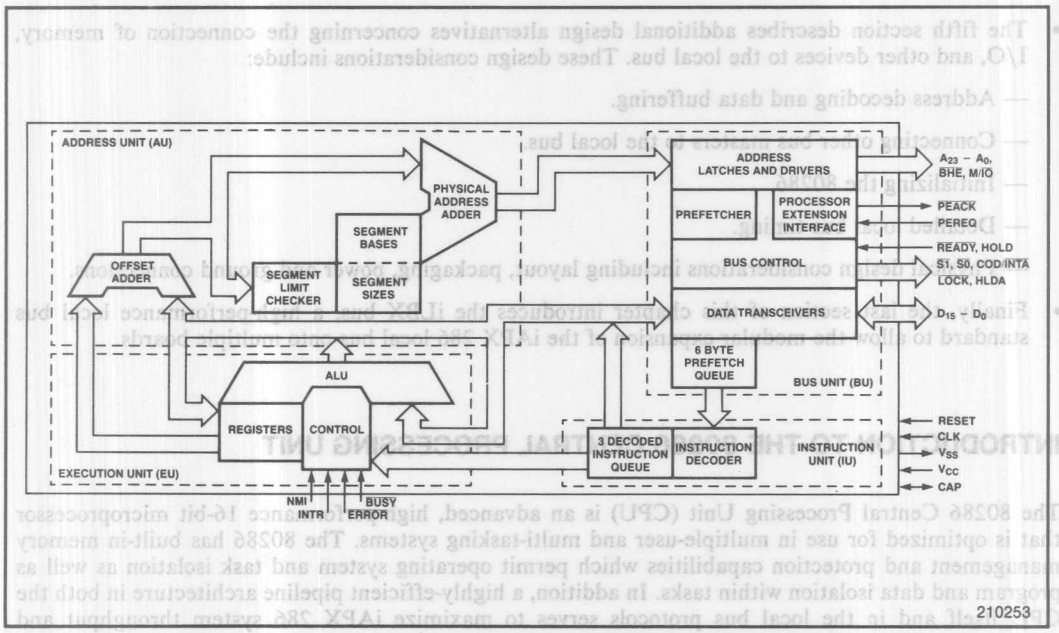


Figure 3-1. 80286 Internal Block Diagram

THE EXECUTION UNIT

Decoded instructions from the Instruction Unit are fetched by the Execution Unit (EU) and executed. The EU uses the Bus Unit to perform data transfers to or from memory and I/O.

THE ADDRESS UNIT

The Address Unit (AU) provides the memory management and protection services for the CPU and translates logical addresses into physical addresses for use by the Bus Unit. A register cache in the AU contains the information used to perform the various memory translation and protection checks for each bus cycle.

THE EFFECTS OF PIPELINING

The four 80286 processing units operate independently and in parallel with one another, overlapping instruction fetches, decoding, and execution to maximize processor throughput and bus utilization. Figure 3-2 shows how this pipelined architecture results in substantially increased performance relative to a processor that fetches and executes instructions sequentially.

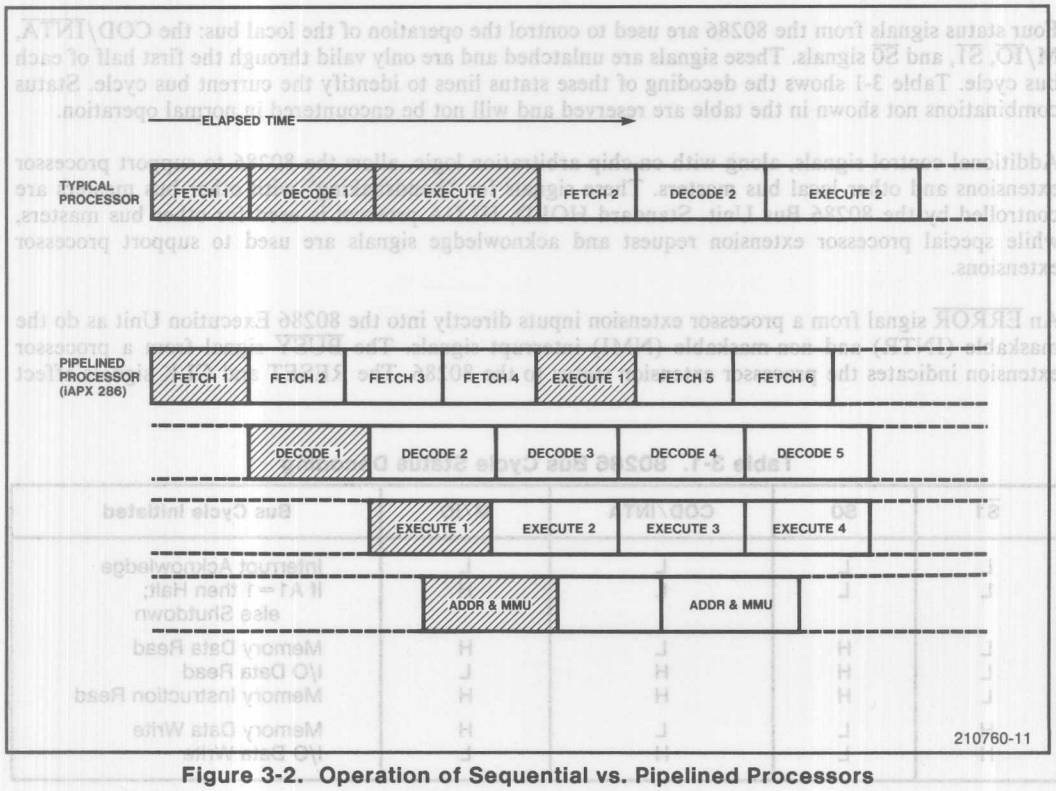


Figure 3-2. Operation of Sequential vs. Pipelined Processors

The iAPX 286 processor operates in one of two modes: iAPX 86 Real-Address mode, and Protected Virtual-Address mode. In both modes, the iAPX 286 executes a superset of the iAPX 86 instruction set.

In iAPX 86 Real-Address mode, the 80286 addresses up to one megabyte of address space using a twenty-bit physical address. In Protected Virtual-Address mode, programs executing on the 80286 can address up to a gigabyte of virtual address space, which the 80286 automatically maps into 16 megabytes of physical address space using a twenty-four-bit physical address.

From a hardware standpoint, these two operating modes differ only in that the upper four address lines (A_{23} through A_{20}) must be ignored when the iAPX 286 is operating in Real-Address mode, and must be decoded along with the other address lines when the iAPX 286 is operating in Protected Virtual-Address mode.

The 80286 Bus Interface

The 80286 CPU connects to external memory, I/O, and other devices using a parallel bus interface. This bus interface consists of a 24-bit address bus, a separate 16-bit data bus, and a number of control and status lines to control the transfer of information across the bus. Taken together, and with additional control signals derived from this basic set, these address, data, and control signals form the basis for the iAPX 286 local bus.

Four status signals from the 80286 are used to control the operation of the local bus: the COD/$\overline{\text{INTA}}$, M/$\overline{\text{IO}}$, $\overline{\text{S1}}$, and $\overline{\text{S0}}$ signals. These signals are unlatched and are only valid through the first half of each bus cycle. Table 3-1 shows the decoding of these status lines to identify the current bus cycle. Status combinations not shown in the table are reserved and will not be encountered in normal operation.

Additional control signals, along with on-chip arbitration logic, allow the 80286 to support processor extensions and other local bus masters. These signals for "handshaking" with other bus masters are controlled by the 80286 Bus Unit. Standard HOLD, HLDA protocol is used for other bus masters, while special processor extension request and acknowledge signals are used to support processor extensions.

An $\overline{\text{ERROR}}$ signal from a processor extension inputs directly into the 80286 Execution Unit as do the maskable (INTR) and non-maskable (NMI) interrupt signals. The $\overline{\text{BUSY}}$ signal from a processor extension indicates the processor extension status to the 80286. The RESET and CLK signals affect

Table 3-1. 80286 Bus Cycle Status Decoding

$\overline{\text{S1}}$	$\overline{\text{S0}}$	COD/$\overline{\text{INTA}}$	M/$\overline{\text{IO}}$	Bus Cycle Initiated
L	L	L	L	Interrupt Acknowledge
L	L	L	H	If A1=1 then Halt; else Shutdown
L	H	L	H	Memory Data Read
L	H	H	L	I/O Data Read
L	H	H	H	Memory Instruction Read
H	L	L	H	Memory Data Write
H	L	H	L	I/O Data Write

the 80286 processor as a whole; these signals do not interface to the device through a specific processing unit. A complete description of the 80286 pin connections can be found in the Appendices.

The following sections describe the iAPX 286 local bus interface in greater detail. First, an overview of the iAPX 286 local bus is given, explaining the relevant concepts and relationships that are involved. This general description is followed by a discussion of specific issues that must be considered when designing an iAPX 286 system.

LOCAL BUS OVERVIEW

The iAPX 286 local bus connects memory and I/O resources to the 80286 processor, using 24 separate address lines, 16 data lines, and a number of status and control signals. Together, these address, data, and control signals allow the 80286 processor to fetch and execute instructions, to manipulate information from both memory and I/O devices, and to respond to interrupts, processor extension requests, and requests from other bus masters.

In many respects, the principles and protocols used in the iAPX 286 local bus are similar to those commonly used in other parallel bus systems. In some respects, however, the iAPX 286 local bus is a high-performance bus that differs somewhat from typical bus patterns. The following sections describe how the iAPX 286 local bus is organized and how it operates to assure the efficient transmission of information between all of the devices on the bus.

Organization of Physical Memory and I/O

The principal use of the local bus is to connect the iAPX 286 processor to memory and I/O devices. When operating in Real-Address mode, the iAPX 286 can directly address up to 1 megabyte of physical memory, while in Protected Virtual-Address mode, the iAPX 286 can address up to 16 megabytes of physical memory. These two operating modes of the 80286 have already been described. Except for the differences in memory size, the organization of memory and I/O is identical for both Real-Address mode and Protected Virtual-Address mode.

In addition to its memory-addressing capabilities, the iAPX 286 can also directly address up to 65,536 8-bit I/O ports or up to 32,768 16-bit I/O ports mapped into a separate I/O address space, in either operating mode. Figure 3-3 illustrates these separate memory and I/O address spaces.

MEMORY ORGANIZATION

The programmer views the memory address space of the iAPX 286 as a sequence of (8-bit) bytes in which any byte may contain an 8-bit data element and any two consecutive bytes may contain a 16-bit (word) data element. Both byte and word information can be assigned to either even or odd addresses—there is no constraint on word boundaries.

As shown in Figure 3-4, the address space is physically implemented on a 16-bit data bus by dividing the address space into two banks of up to 512K bytes (in Real-Address mode) or 8 Mbytes (in Protected mode). The lower half of the data bus (D7-D0) is connected to one bank of memory, and accesses even-addressed bytes (A0=0). The upper half of the data bus (D15-D8) connects to the other bank and contains odd-addressed bytes (A0=1). Address line A0 and Bus High Enable ($\overline{BHE}$) enable the appropriate banks of memory, while the remaining address lines select a specific byte within each bank.

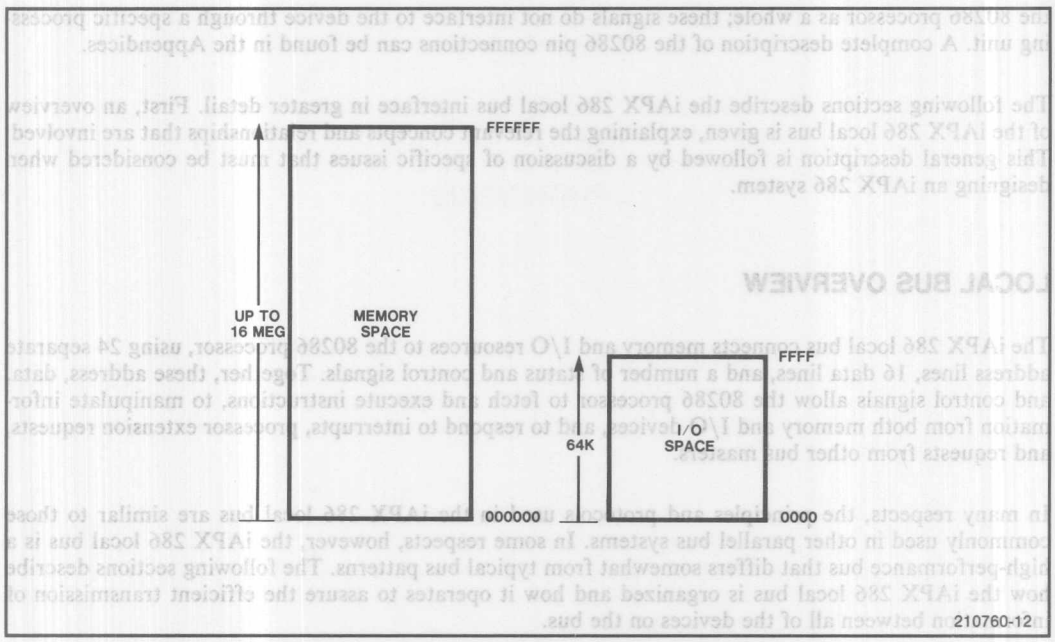


Figure 3-3. Separate Memory and I/O Spaces

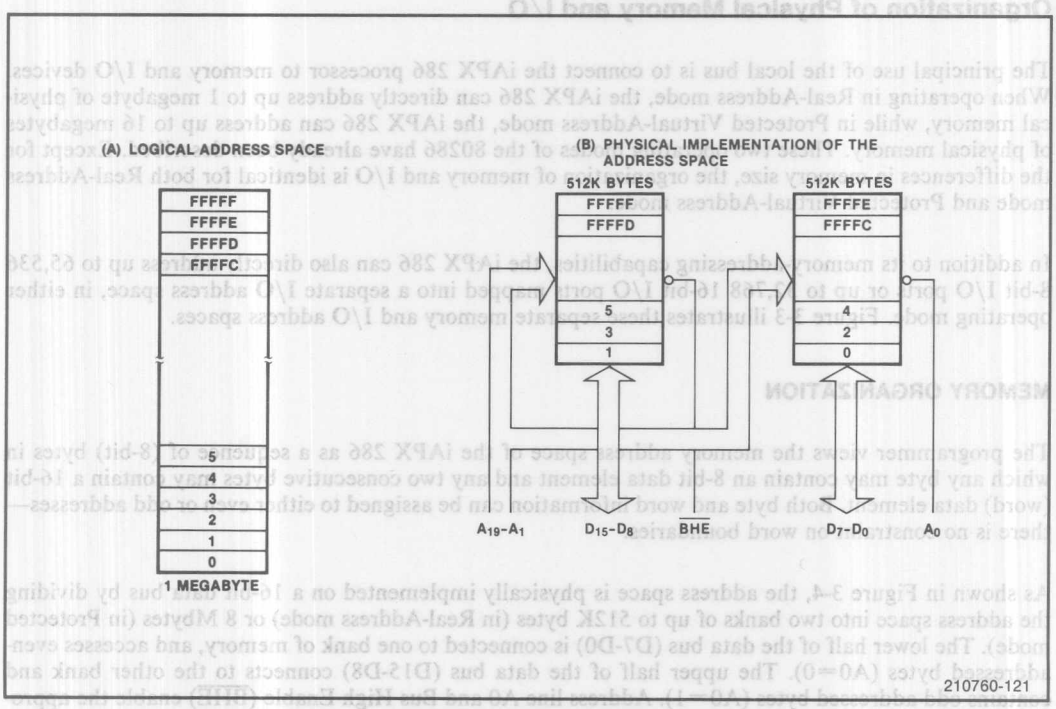


Figure 3-4. iAPX 286 Memory Organization

(When operating in Real-Address mode, address lines A23-A20 should be ignored, and address lines A19-A1 select the specific byte.)

To perform byte transfers to even addresses (Figure 3-5A), the CPU transfers information over the lower half of the data bus (D7-D0). A0 enables the bank connected to the lower half of the data bus to participate in the transfer. Bus High Enable ($\overline{BHE}$ active low), disables the bank on the upper half of the data bus from participating in the transfer. Disabling the upper bank is necessary to prevent a write operation to the lower bank from destroying data in the upper bank. During the transfer, the upper half of the data bus remains in the tri-state OFF condition.

To perform byte transfers to odd addresses (Figure 3-5B), the CPU transfers information over the upper half of the data bus (D15-D8) while $\overline{BHE}$ (active low) enables the upper bank and A0 disables the lower bank. The lower half of the data bus remains in the tri-state OFF condition.

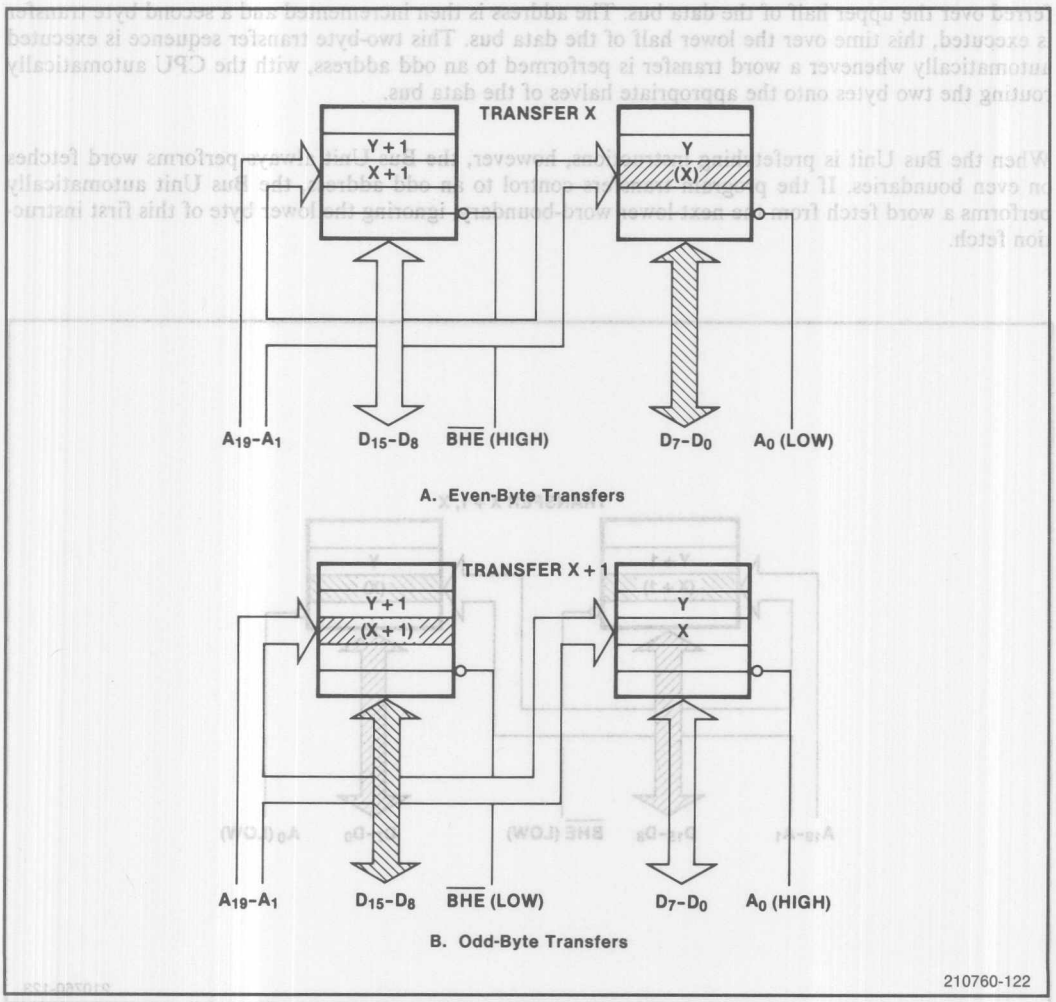


Figure 3-5. 80286 Byte Transfers

byte from an odd-addressed memory location into the CL register (lower half of the CX register) illustrates this routing. The data is transferred into the 80286 over the upper half of the data bus, then automatically redirected to the lower half of the 80286 internal data path and stored into the CL register. This automatic routing ability also allows byte-I/O transfers between the AL register and 8-bit I/O devices that are connected to either half of the 16-bit data bus.

To access even-addressed 16-bit words (two consecutive bytes with the least significant byte at an even byte address), A0 (low) and $\overline{BHE}$ (low) enable both banks simultaneously, while the remaining address lines select the appropriate byte from each bank. Figure 3-6 illustrates this operation.

To access 16-bit word data beginning at an odd address, the 80286 automatically performs two byte accesses. As shown in Figure 3-7, the least significant byte addressed by the address lines is first transferred over the upper half of the data bus. The address is then incremented and a second byte transfer is executed, this time over the lower half of the data bus. This two-byte transfer sequence is executed automatically whenever a word transfer is performed to an odd address, with the CPU automatically routing the two bytes onto the appropriate halves of the data bus.

When the Bus Unit is prefetching instructions, however, the Bus Unit always performs word fetches on even boundaries. If the program transfers control to an odd address, the Bus Unit automatically performs a word fetch from the next-lower word-boundary, ignoring the lower byte of this first instruction fetch.

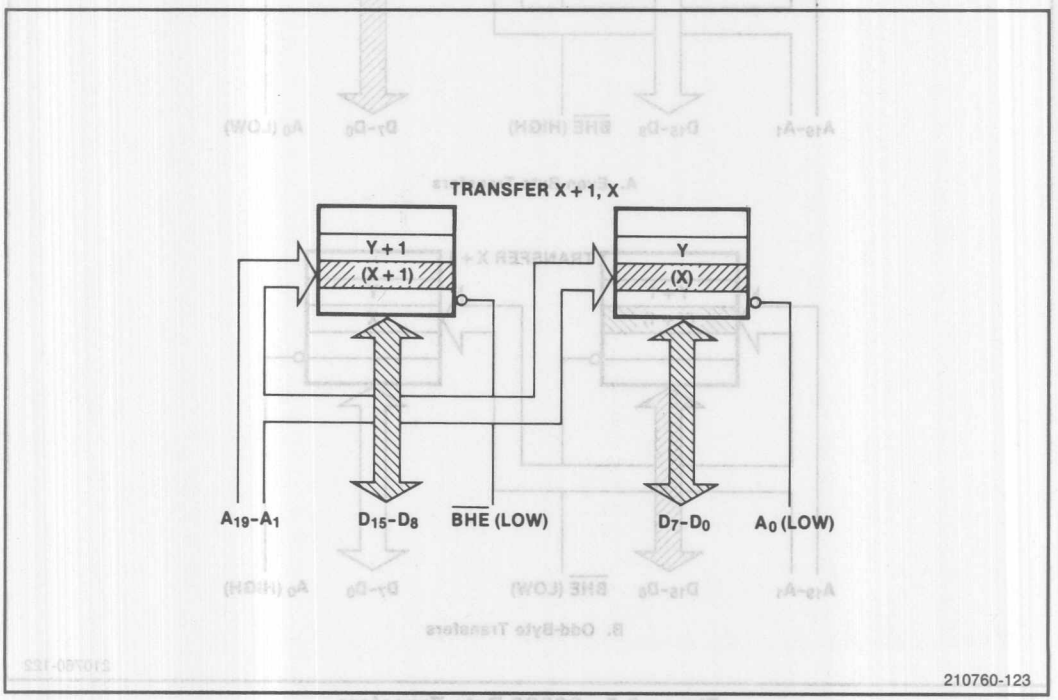


Figure 3-6. Even-Addressed Word Transfer

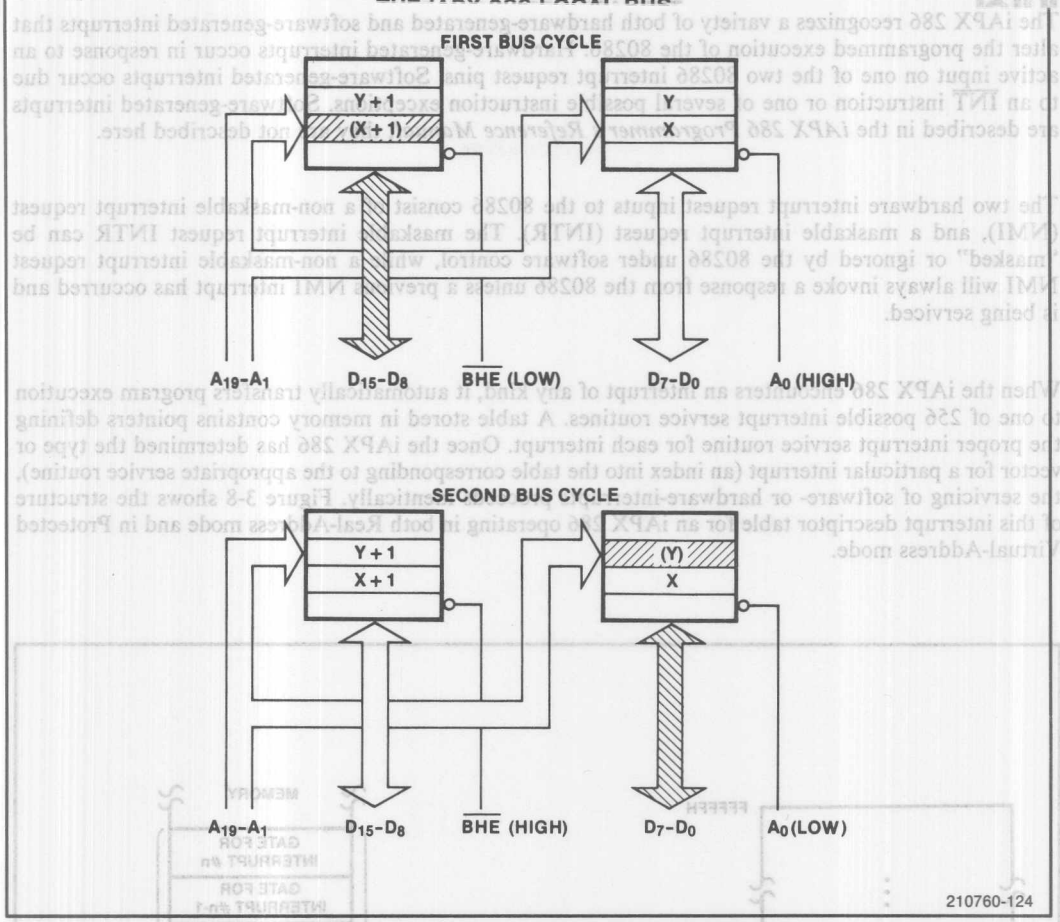


Figure 3-7. Odd-Addressed Word Transfer

I/O ORGANIZATION

The 80286 is capable of interfacing with 8- and 16-bit I/O devices that are mapped into a separate 64K I/O space (memory-mapped I/O is also supported, and is described in Chapter Six). The iAPX 286 I/O space is organized as 65,536 8-bit ports or 32,768 16-bit ports or some combination of the two.

For 8-bit I/O devices, ports on the upper half of the data bus, have odd I/O addresses, while ports on the lower half of the data bus have even I/O addresses. To access an 8-bit I/O device, A0 and $\overline{BHE}$ select the appropriate half of the data bus, and address lines A15-A1 select one of the 32K I/O addresses. During any I/O transfer, address lines A23-A20 are always low.

Sixteen-bit I/O devices always have even addresses to permit the CPU to select and access the entire 16-bit port in a single operation. To access a 16-bit I/O device, address lines A15-A1 select the particular I/O address, while A0 and $\overline{BHE}$ condition the chip select to ensure that a 16-bit transfer is being performed.

Interrupt Organization

The iAPX 286 recognizes a variety of both hardware-generated and software-generated interrupts that alter the programmed execution of the 80286. Hardware-generated interrupts occur in response to an active input on one of the two 80286 interrupt request pins. Software-generated interrupts occur due to an $\overline{\text{INT}}$ instruction or one of several possible instruction exceptions. Software-generated interrupts are described in the *iAPX 286 Programmer's Reference Manual*; they are not described here.

The two hardware interrupt request inputs to the 80286 consist of a non-maskable interrupt request (NMI), and a maskable interrupt request (INTR). The maskable interrupt request INTR can be "masked" or ignored by the 80286 under software control, while a non-maskable interrupt request NMI will always invoke a response from the 80286 unless a previous NMI interrupt has occurred and is being serviced.

When the iAPX 286 encounters an interrupt of any kind, it automatically transfers program execution to one of 256 possible interrupt service routines. A table stored in memory contains pointers defining the proper interrupt service routine for each interrupt. Once the iAPX 286 has determined the type or vector for a particular interrupt (an index into the table corresponding to the appropriate service routine), the servicing of software- or hardware-interrupts proceeds identically. Figure 3-8 shows the structure of this interrupt descriptor table for an iAPX 286 operating in both Real-Address mode and in Protected Virtual-Address mode.

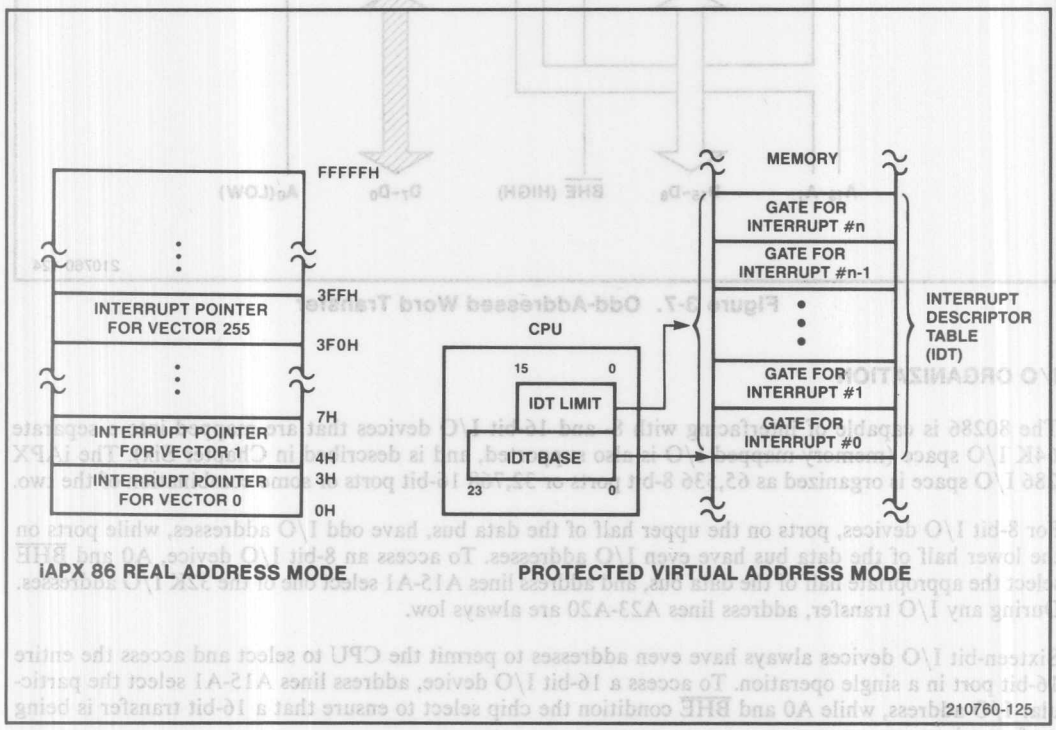


Figure 3-8. Interrupt Descriptor Table Definition

INTERRUPT PRIORITIES

When an interrupt or exception occurs, the iAPX 286 automatically transfers program control to the appropriate interrupt service routine, even if the iAPX 286 is in the middle of processing a previous interrupt. In this manner, the last interrupt processed is the first one serviced.

The single exception to this rule is the 80286 treatment of the INTR interrupt input. When operating in Real-Address mode, any interrupt or exception automatically causes the 80286 to mask any INTR requests for the duration of the interrupt service routine until the occurrence of an IRET instruction or until the service routine explicitly enables INTR interrupts. When the 80286 is operating in Protected mode, the individual task gate for each interrupt service routine specifies whether INTR interrupts are to be masked for the duration of the interrupt service routine.

In the case of simultaneous interrupt or exception requests, the interrupts are processed in the fixed order shown in Table 3-2. Interrupt processing involves saving the flags and the return address, and setting CS:IP to point to the first instruction of the interrupt service routine. If other interrupt requests remain pending, they are processed before the first instruction of the current interrupt handler is executed. In this way, the last interrupt processed is the first one serviced.

To illustrate the information contained in the table, consider what occurs when both an NMI (non-maskable interrupt) request and an INTR (maskable interrupt) request are received simultaneously by an 80286 operating in Real-Address mode. The interrupts are handled in the following order:

1. The NMI interrupt will be processed first, saving the current status and pointing to the first instruction of the NMI service routine.
2. The INTR interrupt request will be masked, since the NMI interrupt causes the 80286 to mask INTR interrupts.
3. If no other exceptions occur, the NMI interrupt service routine will be executed until it is completed.
4. After the NMI service routine has completed, an IRET instruction will be executed to exit from the service routine. This IRET instruction re-enables INTR interrupts.
5. If the INTR interrupt request line is still active, the processor will then respond to the interrupt with two interrupt-acknowledge bus cycles, and then proceed to execute the appropriate interrupt service routine. Any subsequent NMI interrupts that occur during the execution of this service routine will immediately be serviced before execution of the INTR service routine continues.
6. Finally, after all interrupts have been serviced, program control will revert to the original interrupted program.

Table 3-2. Processing Order for Simultaneous Interrupts

Processing Order	Interrupt Source
1 (first)	Processor Exception
2	Single Step
3	NMI (Non-Maskable Interrupt Request)
4	Processor Extension Segment Overrun
5	INTR (Maskable Interrupt Request)
6 (last)	INT Instruction

The following sections describe the iAPX 286 system's different responses to the two hardware-generated interrupt requests, NMI and INTR.

NON-MASKABLE INTERRUPT REQUEST (NMI)

The non-maskable interrupt request (NMI) input to the 80286 is edge-triggered (on a low-to-high transition) and is generally used to signal the CPU of a "catastrophic" event such as the imminent loss of power, memory error, or bus-parity error.

Interrupt requests arriving on the NMI pin are latched by the CPU and cannot be disabled. The NMI input is asynchronous and, in order to ensure that it is recognized, required to have been low for four system clock cycles before the transition and to remain high for a minimum of four additional clock cycles. To guarantee that the NMI input is recognized on a particular clock transition, the required set-up time for NMI is shown in Figure 3-9.

Once it is recognized by the 80286, the non-maskable interrupt automatically causes the 80286 CPU to transfer control to the service routine corresponding to interrupt type 2; this interrupt vector is internally-supplied and generates no external interrupt-acknowledge sequence.

The total time required to service an NMI interrupt is determined by several factors. The total processing time includes:

- The time the interrupt was waiting to be recognized. While servicing one NMI, the 80286 will not acknowledge another NMI until an IRET instruction is executed.
- Additional time allowed for the completion of the instruction currently being executed by the 80286. These instructions could be IRET, MUL, or task-switch instructions.
- Time required for saving the CS, IP, and Flags registers.
- Any time required by the interrupt service routine to save the contents of additional registers not automatically saved by the 80286.

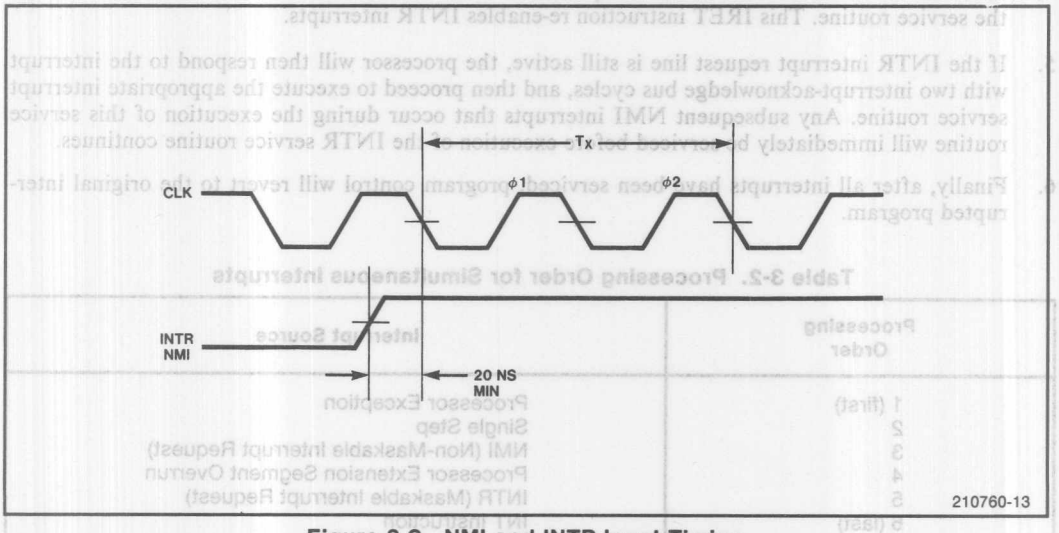


Figure 3-9. NMI and INTR Input Timing

Since the NMI interrupt will be recognized only on an instruction boundary, the time required for the CPU to recognize the request (interrupt latency) depends on how many clock periods remain in the execution of the current instruction.

On the average, the longest latency can occur if the interrupt request arrives while task-switch is being executed in Protected mode. Other instructions resulting in long interrupt latencies are multiplications and divisions. In the case of an instruction that loads the Stack Segment register, the interrupt will not be recognized until after the following instruction is executed, to allow loading of the entire stack pointer without intervening interrupts.

While executing an NMI service routine, the 80286 will not service additional NMI interrupts or processor extension segment-overrun interrupts until an interrupt return (IRET) instruction is executed or the CPU is reset. If an NMI interrupt request occurs while the 80286 is servicing a prior NMI request, the occurrence of the second NMI request will be saved and the request will be serviced after the CPU executes an IRET instruction.

During the NMI service routine, an 80286 operating in Real-Address mode automatically masks any INTR interrupt requests. In Protected mode, the NMI service routine is entered through either an interrupt gate or a task gate. The new task context may be defined with INTR interrupts either enabled or disabled. To re-enable INTR interrupts during the NMI service routine, the 80286 Interrupt (IF) flag can be set true. In any case, the IF flag is automatically restored to its original state when the service routine is completed (when the Flags register is restored following the first IRET instruction).

MASKABLE INTERRUPT REQUEST (INTR)

The INTR (Interrupt Request) line allows external devices to interrupt 80286 program execution. INTR is usually driven by an Intel 8259A Programmable Interrupt Controller (PIC), which, in turn, is connected to devices that require interrupt servicing.

The 8259A is a very flexible device that is controlled by software commands from the 80286 (the 8259A PIC appears to the CPU as a set of I/O ports). Its main job is to accept interrupt requests from devices connected to it, to determine which request has the highest priority, and then activate the INTR line to interrupt the CPU and supply an appropriate interrupt vector.

Figure 3-10 shows a block diagram of a multiple 8259A subsystem that uses a master interrupt controller driven by slave interrupt controllers. Chapter Five contains detailed information on interfacing the 8259A to an iAPX 286 system.

The INTR input to the 80286 is level-sensitive and can be asynchronous to the system clock. INTR must be active for at least two processor clock cycles before the current instruction ends in order to interrupt before the next instruction begins. When the 80286's Interrupt Flag (IF) is enabled and the INTR signal is driven active high, the 80286 will acknowledge the interrupt on the next instruction boundary. To guarantee recognition on a particular clock cycle, INTR must be set-up a minimum of 20 ns before the start of the clock cycle. INTR should remain active until the first INTA bus cycle to guarantee that the CPU responds to the interrupt. Figure 3-9 shows the required timing of the INTR input.

The iAPX 286 acknowledges an INTR interrupt request by executing two interrupt-acknowledge (INTA) bus cycles (INTA bus cycles are described in a later section). The first INTA cycle signals all 8259As that the interrupt request has been honored. During the second INTA cycle, the 8259A with the highest-priority interrupt pending responds by placing a byte containing the interrupt type (0-255) onto the data bus to be read by the 80286. This interrupt type must have been previously programmed into the 8259A to correspond to the service routine for the particular device requesting service.

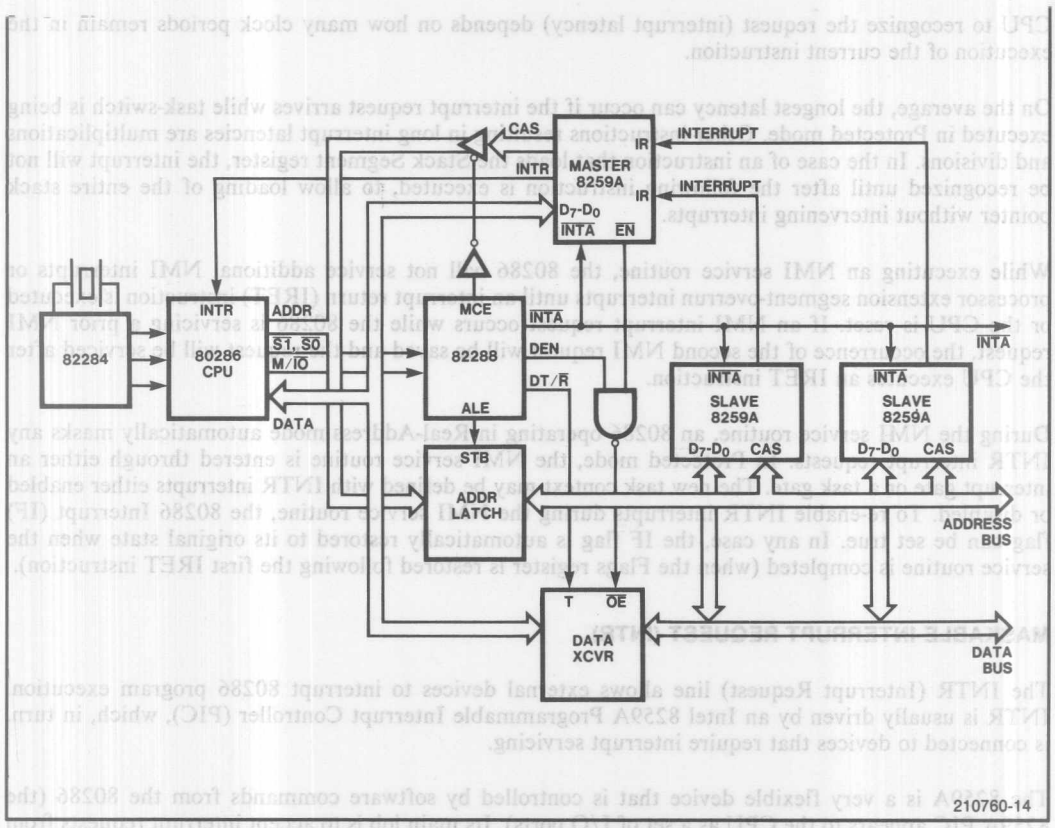


Figure 3-10. 8259A PIC Driving 80286 INTR Input

Following the second INTA cycle, the 80286 automatically transfers control to the interrupt service routine corresponding to that particular interrupt.

During the interrupt-acknowledge cycles, any bus hold requests arriving via the 80286 HOLD line are not honored until the interrupt-acknowledge cycles have been completed. In addition, the CPU activates its LOCK signal during the first of these cycles to prevent other bus masters from taking control of the bus.

As for the NMI interrupt, the total time required to service an INTR interrupt is determined by the type of instructions used and how long interrupts are disabled. The total servicing time is comprised of the same elements as described for the NMI interrupt.

Since INTR requests will be acknowledged only by the 80286 on an instruction boundary, the time required for the CPU to recognize the request (interrupt latency) depends on how many clock periods remain in the execution of the current instruction.

The same factors affecting interrupt latency for the NMI interrupt affect the latency of the INTR interrupt request. Specific instructions resulting in long interrupt latencies are multiplications, divisions, task switches in Protected mode, or instructions that load the Stack Segment register, as described for the NMI interrupt.

Pipelined Address Timing

The iAPX 286 local bus differs most from a typical microprocessor bus in its use of pipelined address timing. To achieve high bus throughput, the pipelined address timing used by the iAPX 286 allows overlapped bus cycles when accessing memory and I/O. The resulting increase in bus throughput is achieved without requiring a proportional increase in memory speed.

This pipelined timing differs from that of a typical microprocessor bus cycle. During transfers for typical processors, an address is transmitted at the start of the bus cycle and held valid during the entire cycle.

Using pipelined timing, the 80286 places the address for the next memory or I/O operation on the bus even before the previous bus operation has completed. This overlapping (pipelining) of successive bus operations permits the maximum address setup time before data is required by the CPU or memory.

Figure 3-11 illustrates how this address pipelining results in improved bus throughput, even though the address and data setup times for individual memory operations are identical for the two examples shown. For an 8-MHz iAPX 286 system executing word transfers with zero wait states, data can be transferred at the rate of 8 Megabytes per second while still allowing individual address access times of 242 ns.

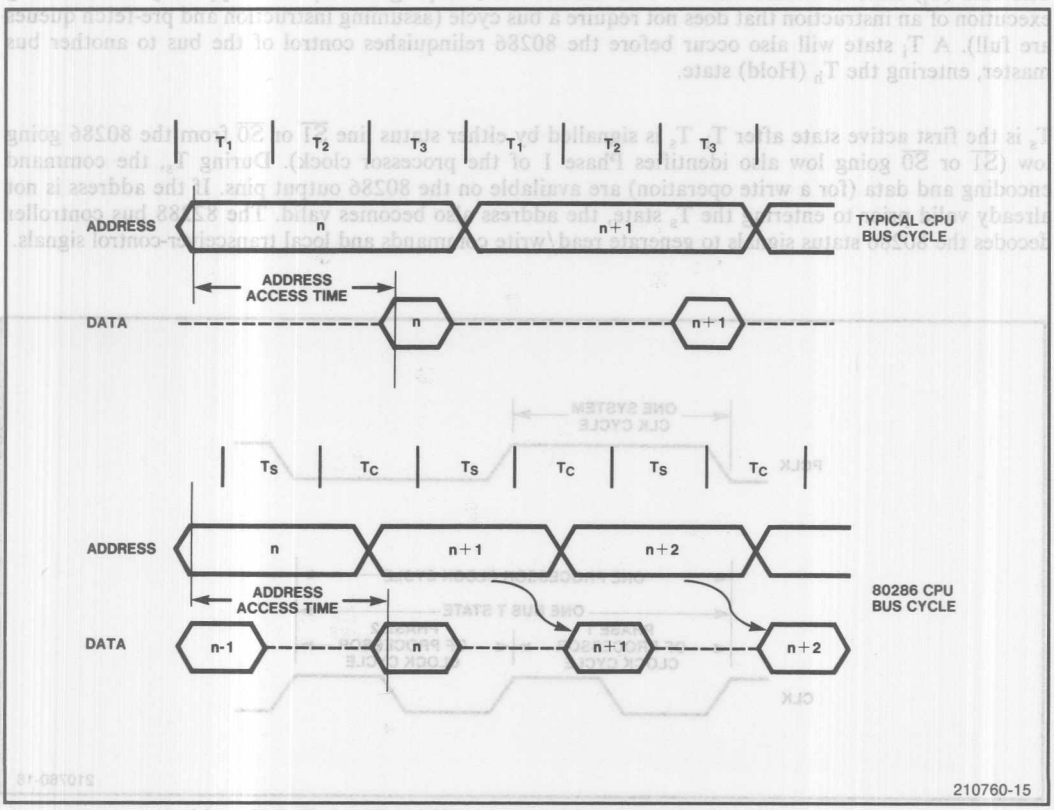


Figure 3-11. Typical and Pipelined Bus Operations

In the following sections, the complete iAPX 286 bus cycle is described, and the implementation of pipelined timing is shown in greater detail. In a later section of this chapter, specific design considerations and components are described for building high-performance iAPX 286 systems that take advantage of this pipelined address timing.

iAPX 286 Local Bus States

The 80286 CPU uses a double-frequency system clock (CLK) to control bus timing. The CPU internally divides the system CLK by two to produce the internal processor clock, which determines the local bus state. Each processor clock cycle is composed of two system CLK cycles, called Phase 1 and Phase 2. The 82284 Clock Generator produces a processor clock output (PCLK) that identifies the phase of the internal processor clock. Figure 3-12 shows the relationship between the system CLK and the processor clock (PCLK).

The iAPX 286 local bus has three basic states: Idle (T_i), Send-Status (T_s), and Perform-Command (T_c). The 80286 CPU also has a fourth state called Hold (T_h). A T_h state indicates that the 80286 has surrendered control of the local bus to another bus master in response to a HOLD request. Each bus state is one processo clock long. Figure 3-13 shows the 80286 CPU states and the allowed transitions.

The Idle (T_i) state indicates that no data transfers are in progress. T_i states typically occur during execution of an instruction that does not require a bus cycle (assuming instruction and pre-fetch queues are full). A T_i state will also occur before the 80286 relinquishes control of the bus to another bus master, entering the T_h (Hold) state.

T_s is the first active state after T_i. T_s is signalled by either status line $\overline{S1}$ or $\overline{S0}$ from the 80286 going low ($\overline{S1}$ or $\overline{S0}$ going low also identifies Phase 1 of the processor clock). During T_s, the command encoding and data (for a write operation) are available on the 80286 output pins. If the address is not already valid prior to entering the T_s state, the address also becomes valid. The 82288 bus controller decodes the 80286 status signals to generate read/write commands and local transceiver-control signals.

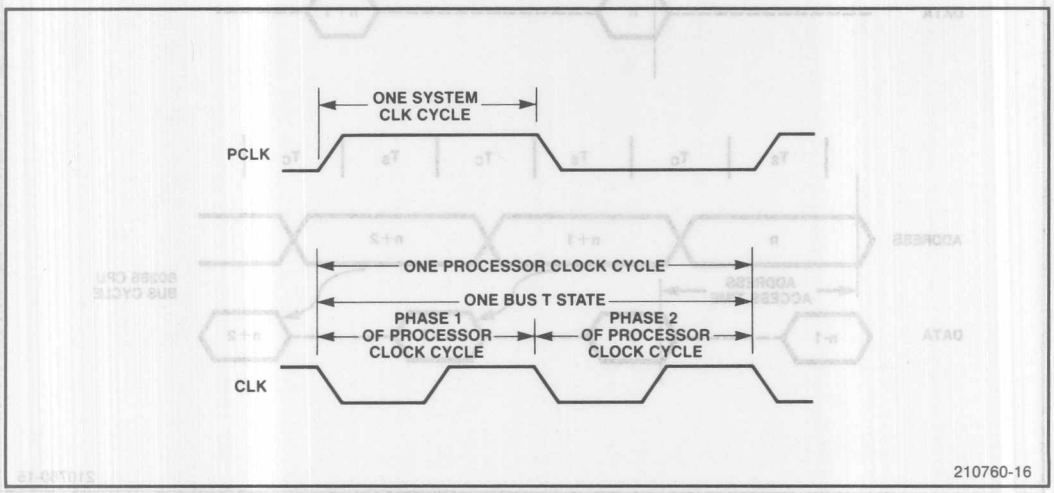


Figure 3-12. System and Processor/PCLK Clock Relationships

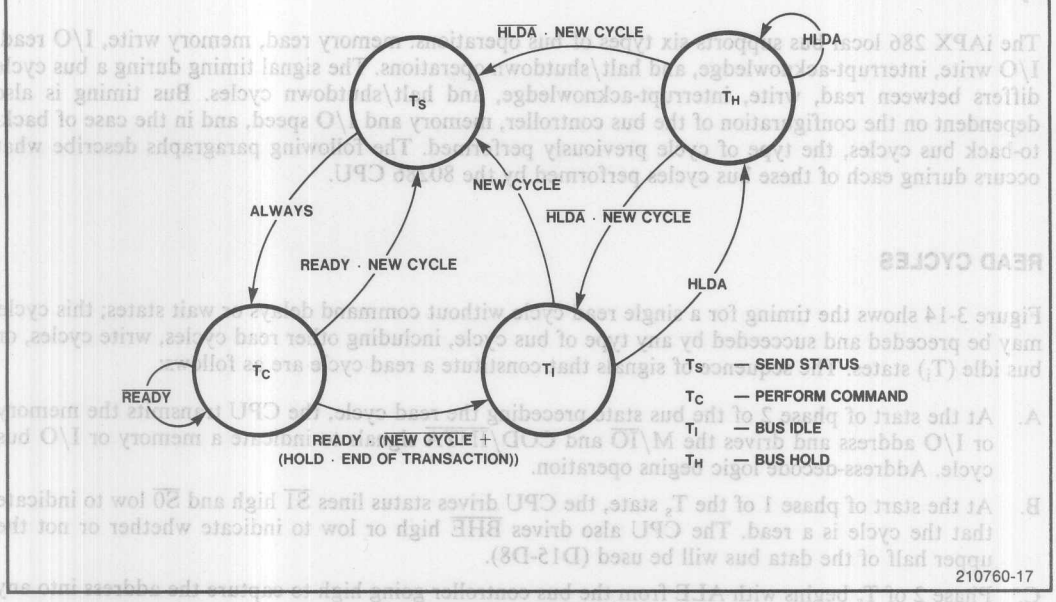


Figure 3-13. 80286 States

After T_s, the perform command (T_c) state is entered. Memory or I/O devices are expected to respond to the bus operation during T_c, either transferring read data to the CPU or accepting write data. The $\overline{READY}$ input to the 80286 either terminates the bus cycle or causes the T_c state to be repeated. T_c states may be repeated as many times as necessary to assure sufficient time for the memory or I/O device to respond.

Following a T_c state, the bus may enter immediately into a T_s state, beginning another bus operation, or may enter the T_i (Idle) state. This ability to immediately execute back-to-back bus operations leads to high bus utilization and contributes to the iAPX 286's high performance.

If another bus master requests and is granted control of the local bus, the 80286 will enter the T_h (Hold) state. If the 80286 is in the T_c state when the HOLD request is received, the bus will pass through one T_i (Idle) state before the 80286 grants control to the requesting bus master. During Hold (T_h), he 80286 floats all address, data, and status output pins, allowing the requesting bus master to control the local bus. The 80286 HLDA output signal acknowledges the HOLD request and indicates that the CPU has entered the T_h state.

iAPX 286 Bus Operations

The 80286 Bus Unit executes bus operations whenever the Execution Unit requires a bus operation as part of an instruction execution, whenever a processor extension requests a bus operation, or when sufficient room becomes available in the instruction pre-fetch queue. When no bus operations are in progress or requested, the local bus remains in the idle state.

Each local bus operation consists of one or more bus cycles. Each bus cycle consists of one Send-Status (T_s) state followed by one or more Perform-Command (T_c) states. At maximum speed, the local bus alternates between the T_s and T_c states, transferring one word of information every two processor clock cycles.

The iAPX 286 local bus supports six types of bus operations: memory read, memory write, I/O read, I/O write, interrupt-acknowledge, and halt/shutdown operations. The signal timing during a bus cycle differs between read, write, interrupt-acknowledge, and halt/shutdown cycles. Bus timing is also dependent on the configuration of the bus controller, memory and I/O speed, and in the case of back-to-back bus cycles, the type of cycle previously performed. The following paragraphs describe what occurs during each of these bus cycles performed by the 80286 CPU.

READ CYCLES

Figure 3-14 shows the timing for a single read cycle without command delays or wait states; this cycle may be preceded and succeeded by any type of bus cycle, including other read cycles, write cycles, or bus idle (T_i) states. The sequence of signals that constitute a read cycle are as follows:

A. At the start of phase 2 of the bus state preceding the read cycle, the CPU transmits the memory or I/O address and drives the M/$\overline{IO}$ and COD/$\overline{INTA}$ signals to indicate a memory or I/O bus cycle. Address-decode logic begins operation.

B. At the start of phase 1 of the T_s state, the CPU drives status lines $\overline{S1}$ high and $\overline{S0}$ low to indicate that the cycle is a read. The CPU also drives $\overline{BHE}$ high or low to indicate whether or not the upper half of the data bus will be used (D15-D8).

C. Phase 2 of T_s begins with ALE from the bus controller going high to capture the address into any external address latches.

D. At the start of the first T_c state, ALE goes low to latch the address and chip selects for the remainder of the bus cycle. The bus controller drives DT/$\overline{R}$ and MRDC low to condition the direction of the data transceivers and enable the memory or I/O location to be read. DEN from the bus controller then goes active (high) to enable the data transceivers. Status ($\overline{S1}$ or $\overline{S0}$) is removed in preparation for the next cycle.

E. At the start of Phase 2 of T_c, the address, M/$\overline{IO}$, and COD/$\overline{INTA}$ lines change to reflect the following bus operation (or enter the tri-state OFF condition if the following bus state is an INTA cycle or a T_i state before bus-hold acknowledge).

F. At the end of T_c, the $\overline{READY}$ signal is sampled. If $\overline{READY}$ is low, the input data is assumed to be valid at the CPU data pins; the CPU reads the data from the bus and the bus controller deactivates MRDC and DEN, and returns DT/$\overline{R}$ to its normal active-high state. If $\overline{READY}$ is sampled high, additional T_c states will be executed until $\overline{READY}$ is sampled low.

WRITE CYCLES

Figure 3-15 shows the timing for a single write cycle without command delays or wait states; the timing of this write cycle is very much the same as for the read cycle just described, although the control signalling is of course different. The sequence of signals that constitute a write cycle are as follows:

A. As in the case of read cycles, the CPU drives the address, M/$\overline{IO}$, and COD/$\overline{INTA}$ signals at the start of phase 2 of the state preceding the bus cycle. Address decode logic begins operating.

B. At the start of T_s, the CPU drives $\overline{S1}$ low and $\overline{S0}$ high to indicate a write cycle. $\overline{BHE}$ is driven high or low to enable or disable the high byte of the data bus (D15-D0).

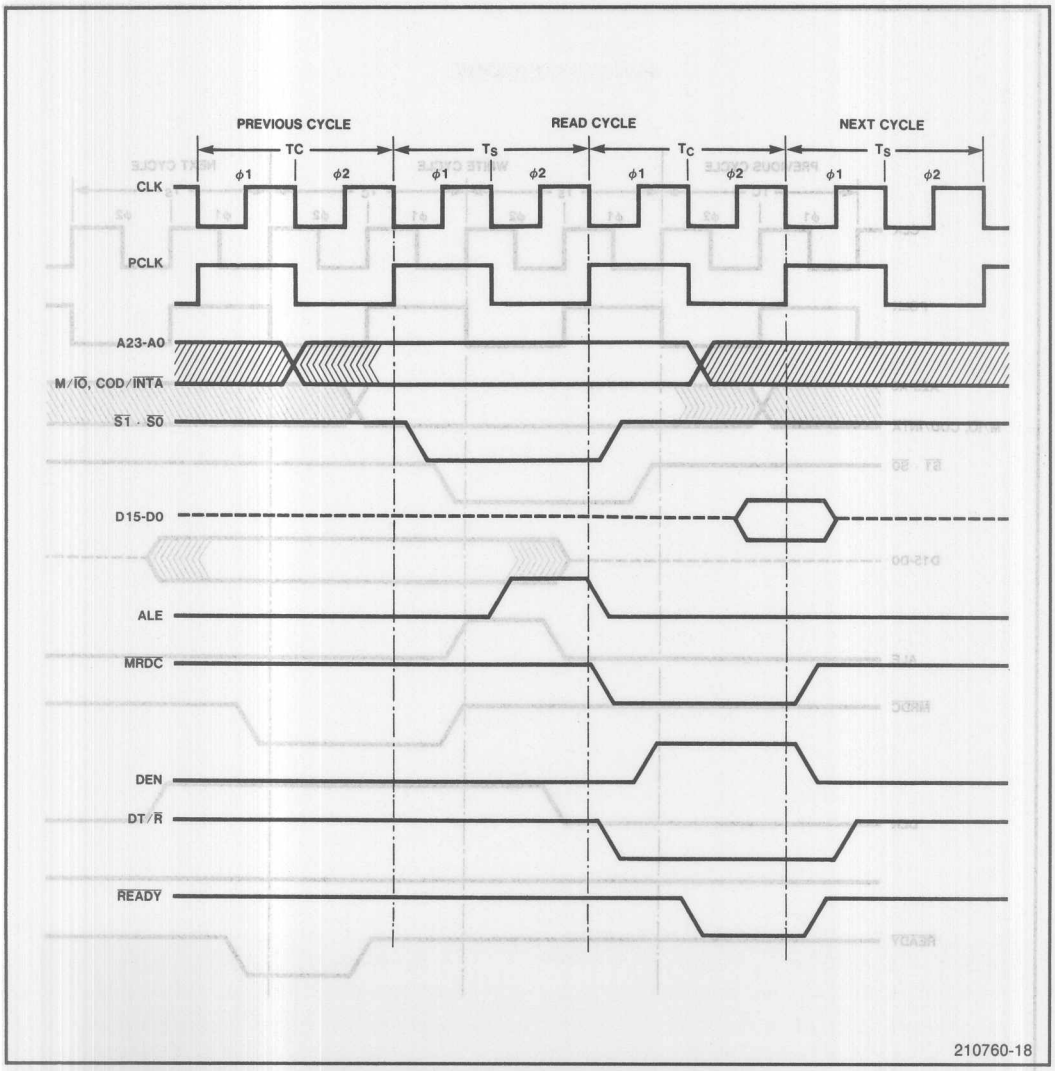


Figure 3-14. iAPX 286 Read Cycle

C. At the start of phase 2 of T_s, the CPU outputs the data to be written to memory or I/O while the bus controller drives ALE and DEN high to capture the address in external address latches and to enable the data transceivers. DT/$\overline{R}$ is normally conditioned to transmit write data through the transceivers; DT/$\overline{R}$ does not change at this time.

D. At the start of the first T_c, the CPU removes its status signals. The bus controller drives MWTC and ALE low to enable a memory or I/O write and to latch the address and chip selects for the rest of the bus cycle.

E. At the start of Phase 2 of T_c, the address, M/$\overline{IO}$, and COD/$\overline{INTA}$ lines change to reflect the following bus operation (or enter the tri-state OFF condition if the following bus state is an INTA cycle or a T_i state before bus-hold acknowledge).

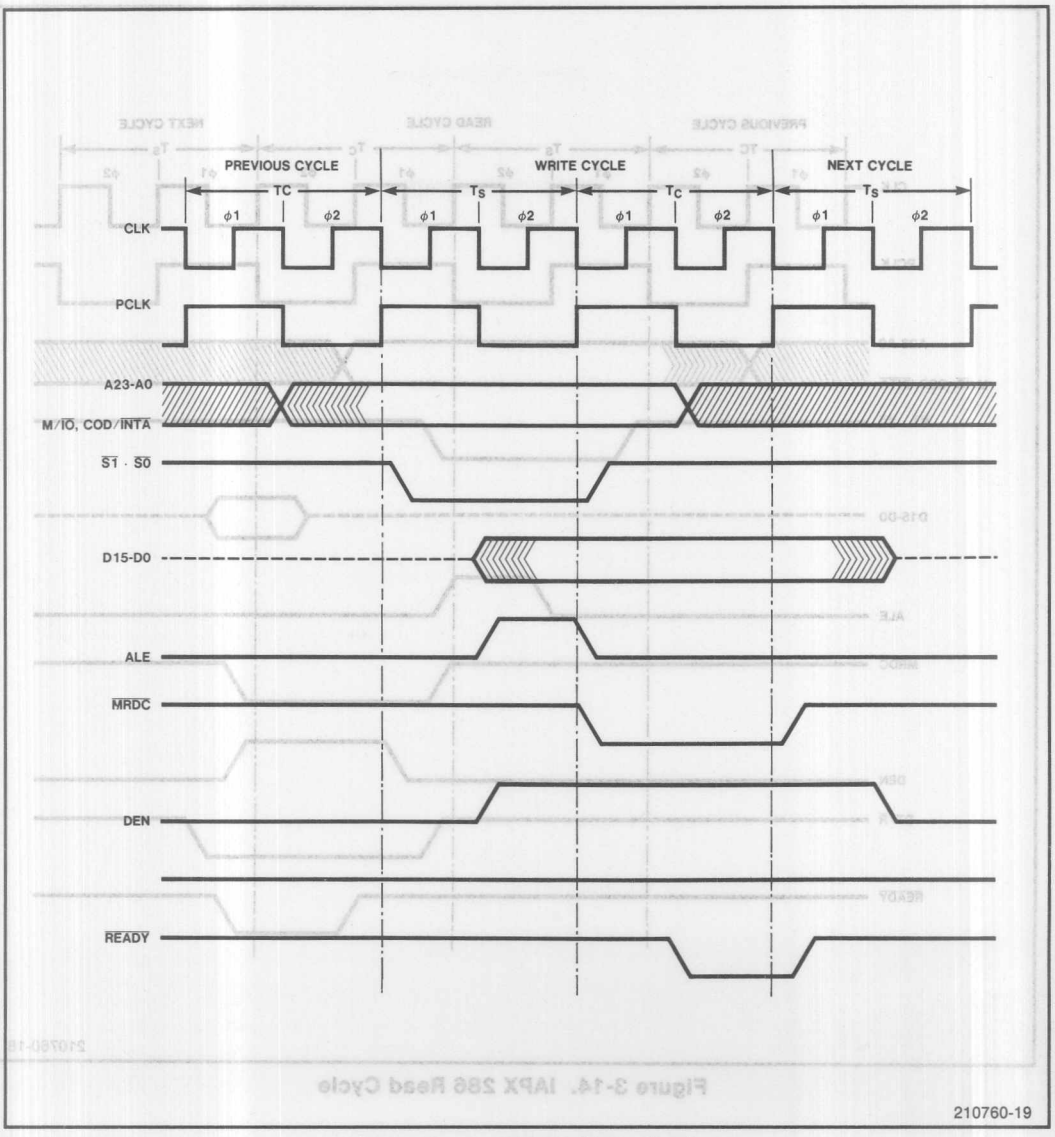


Figure 3-15. 80286 Write Cycle

F. At the end of T_c, the $\overline{\text{READY}}$ input is sampled. If $\overline{\text{READY}}$ is low, the bus controller drives MWTC high to latch data into the selected memory or I/O device. If $\overline{\text{READY}}$ is sampled high, additional T_c states are executed until $\overline{\text{READY}}$ is sampled low.

G. At the start of Phase 2 of the following T_s or T_i bus state, the DEN signal is disabled, and the data bus may change to reflect either new write data or enter the tri-state OFF condition. In this way, write data is held active well into the following bus cycle to provide sufficient data-hold time following termination of the write command.

INTERRUPT-ACKNOWLEDGE CYCLES

Interrupt-acknowledge cycles are performed by the CPU in response to an external interrupt request asserted via the INTR input pin. After recognizing the external interrupt request, the CPU executes a sequence of two back-to-back interrupt-acknowledge ($\overline{INTA}$) cycles to input an 8-bit vector that identifies the interrupting source and directs the 80286 to an interrupt handling routine. Figure 3-16 shows the sequence of signals that constitute the interrupt-acknowledge sequence:

A. At the start of phase 2 of the state preceding the first interrupt-acknowledge cycle, the address lines enter the tri-state OFF condition and M/$\overline{IO}$ and COD/$\overline{INTA}$ are driven low to identify the coming interrupt-acknowledge cycle.

B. At the start of T_s, $\overline{LOCK}$ and status lines $\overline{S1}$ and $\overline{S0}$ are driven low. $\overline{LOCK}$ prevents another bus master in a multi-master system from gaining control of the bus between the two interrupt-acknowledge ($\overline{INTA}$) cycles.

C. At the start of Phase 2 of T_s, the bus controller drives MCE and ALE high. MCE (Master Cascade Enable) enables the master 8259A interrupt controller (in a system that uses multiple interrupt controllers) to drive the cascade address onto the local address bus for distribution to any slave interrupt controllers.

D. At the start of the first T_c, the CPU drives status lines $\overline{S1}$ and $\overline{S0}$ high. The bus controller drives $\overline{INTA}$, ALE, and DT/$\overline{R}$ low. This first INTA signal freezes the contents of the interrupt controller; ALE latches the cascade address onto the system address bus for use by the slave interrupt controllers; and DT/$\overline{R}$ places the data transceivers into the receiver mode (any data on the data bus is ignored during this first $\overline{INTA}$ cycle). The bus controller drives DEN high to enable the data transceivers.

E. Halfway through the first T_c state, the CPU removes the M/$\overline{IO}$ and COD/$\overline{INTA}$ signals while the bus controller drives MCE low (ALE has already fallen low to latch the cascade address onto the address bus). $\overline{READY}$ is high to force a second T_c state (two T_c states are required to meet the minimum 8259A $\overline{INTA}$ pulse width).

F. At the start of the second T_c state, the CPU drives $\overline{LOCK}$ high. If the system bus was used for the first $\overline{INTA}$ cycle, the 82289 Bus Arbiter will not relinquish the bus until the second $\overline{INTA}$ cycle is complete.

G. $\overline{READY}$ going low at the start of phase 2 of T_c terminates the cycle at the end of T_c. The Bus Controller drives $\overline{INTA}$ and DT/$\overline{R}$ high and DEN low. The CPU drives the address pins out of TRI-STATE off during the second half of the second T_c.

H. Three idle states (T_i) are automatically inserted between the two $\overline{INTA}$ bus cycles to allow for the minimum 8259A $\overline{INTA}$-to-$\overline{INTA}$ time and CAS (Cascade Address) output delay.

I. In terms of signal timing, the second $\overline{INTA}$ cycle is almost identical to the first. $\overline{LOCK}$ remains high during the second cycle, and the interrupt vector is read from the lower half of the data bus at the end of the second T_c state. Two T_c states are required during this second $\overline{INTA}$ cycle to meet the minimum $\overline{INTA}$ pulse width and to prevent contention between the CPU and the cascade address latches that might occur as the CPU drives the address bus when the cascade address is being disabled. The CPU does not start driving the address bus until the second half of the second $\overline{INTA}$ T_c bus state.

HALT/SHUTDOWN CYCLES

The 80286 externally indicates halt and shutdown conditions as a bus operation. These two conditions occur due either to an HLT instruction or multiple protection exceptions while attempting to execute one instruction.

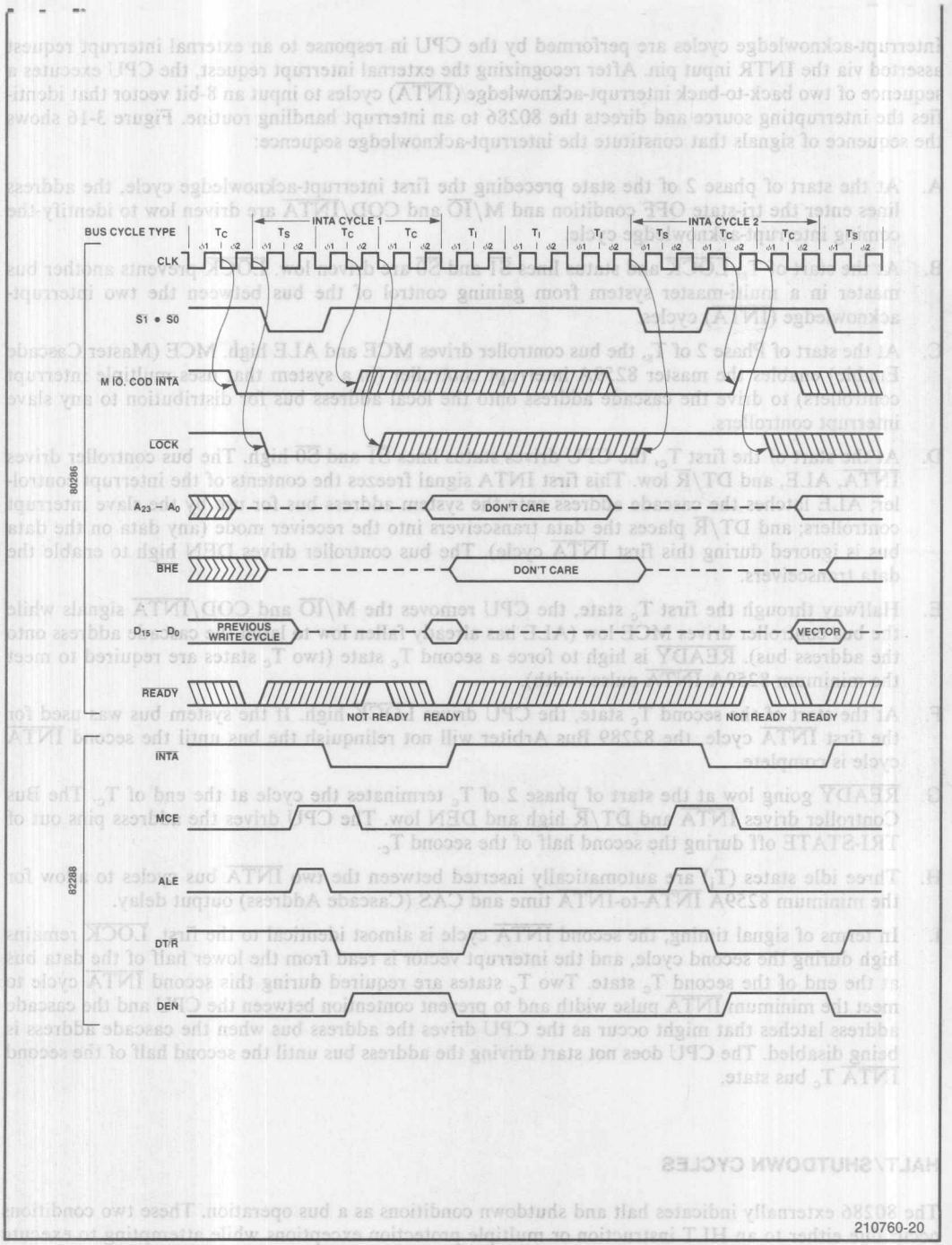


Figure 3-16. Interrupt-Acknowledge Sequence

A halt or shutdown bus operation is signalled when S1, S0, and COD/INTA are low and M/IO is high. The address line A1 distinguishes between a halt or shutdown condition; A1 high indicates a halt condition, and A1 low indicates a shutdown condition. The 82288 does not issue ALE, nor is $\overline{\text{READY}}$ required to terminate a halt or shutdown bus operation.

During a halt condition, the 80286 may service processor extension requests (PEREQ) or HOLD requests. The 80286 will remain in the halt state until an NMI, RESET, processor extension segment overrun exception, or INTR interrupt (if interrupts are enabled) forces the processor out of halt.

During a shutdown condition, the 80286 may similarly service processor extension requests or HOLD requests, except that a processor extension segment overrun exception during shutdown will inhibit further servicing of processor extension requests. Only a non-maskable interrupt (NMI) or RESET can force the 80286 out of the shutdown condition.

iAPX 286 Bus Usage

The previous sections described the characteristics of read and write cycles that can occur on the iAPX 286 local bus. These bus cycles are initiated by the 80286 Bus Unit in order to perform various functions. These functions include:

- prefetching instructions ahead of the current instruction that is executing
- transferring data required by an executing instruction
- transferring data for a processor extension through the processor extension data channel

The following sections describe the rules and conditions that determine which bus operation will be performed next by the 80286. These rules help to better explain the way in which the iAPX 286 uses its local bus.

LOCAL BUS USAGE PRIORITIES

The iAPX 286 local bus is shared between the 80286 internal Bus Unit and external HOLD requests. In the course of performing the three types of bus operations listed above, the 80286 Bus Unit also must contend with transactions that require more than one bus operation to complete. Table 3-3 shows the priorities followed by the 80286 Bus Unit in honoring simultaneous requests for the local bus.

PREFETCH OPERATIONS

The Bus Unit prefetches instructions when the local bus would otherwise be idle. When prefetching instructions, the BU obeys the following general rules:

- A prefetch bus cycle is requested when at least two bytes of the 6-byte prefetch queue are empty.
- Prefetches normally occur to the next consecutive address indicated by the Code Segment Instruction Pointer.
- The prefetcher normally performs word prefetches independent of the byte-alignment of the code segment base in physical memory.
- The prefetcher performs only a byte code-fetch operation for control transfers to an instruction beginning on an odd physical address.
- Prefetching stops whenever a control transfer or HLT instruction is decoded by the IU and placed into the instruction queue.
- In Real-Address mode, the prefetcher may fetch up to six bytes beyond the end of a code segment.

Table 3-3. Local Bus Usage Priorities

Priority	Operations Requesting the Local Bus
1 (Highest)	Any data transfer that asserts $\overline{\text{LOCK}}$ either explicitly (via the $\overline{\text{LOCK}}$ instruction prefix) or implicitly (segment descriptor access, interrupt-acknowledge sequences, or an XCHG with memory).
2	The second of the two byte-transfers required to transfer a word operand at an odd physical address, or the second transfer required to perform a processor extension data channel transfer.
3	A request for the local bus via the HOLD input.
4	A processor extension data channel transfer via the PEREQ input.
5	A data transfer requested by the EU as part of an instruction.
6 (Lowest)	An instruction prefetch performed by the BU. The EU will inhibit prefetching two processor clocks in advance of any data transfers to minimize waiting by the EU for a prefetch to complete.

- In Protected mode, the prefetcher will never cause a segment-overrun exception. The prefetcher stops at the last physical word in the code segment. Exception 13 will occur if the program attempts to execute beyond the last full instruction in the code segment.
- If the last byte of a code segment appears on an even physical memory address, the prefetcher will read the next physical byte of memory (perform a word code fetch). The value of the additional byte is ignored and any attempt to execute it causes exception 13.
- Instruction prefetch operations can occur back-to-back with other prefetches or other bus cycles, with no idle clocks on the bus.

DATA OPERATIONS

The Bus Unit performs data operations when the Execution Unit (EU) needs to read or write data to main memory or I/O. When performing data operations, the Bus Unit obeys the following general rules:

- Since the EU must wait on data reads, data operations have priority over prefetch operations when both data and prefetch operations are ready to begin. A data operations, however, will not interrupt a prefetch operation that has already started.
- If the bus is currently in use when the EU requests a write cycle, the address and data for the write cycle are stored by the Bus Unit in temporary registers until the current bus cycle completes. This allows the EU to begin executing the next instruction without having to wait for a prefetch or other bus operation to finish.
- Only one write operation can be buffered by the Bus Unit. If the EU requests a write cycle while these temporary buffers are full, the EU must wait for the buffers to empty before it can continue executing instructions.
- Any combination of data read and write operations can occur back-to-back with other bus operations, with no idle bus cycles. The string move or I/O instructions in particular transfer data with no idle states on the bus.

DATA CHANNEL TRANSFERS

The 80286 Bus Unit contains a processor extension data channel that supports data transfers between memory and processor extensions like the 80287 Numeric Processor Extension. Transfers are requested by the processor extension and performed by the CPU Bus Unit. Chapter Six describes the iAPX 286 processor extension interface in more detail. The following guidelines govern data channel activity:

- Data channel transfers have priority over prefetch and execution data cycles if all three operations are requested at the same time.
- The transfers require at least two bus cycles to complete. For example, to transfer a word operand to the 80287, the CPU performs the first cycle to read the operand from memory. A second transfer writes the word to the 80287.
- The bus timing of these transfers is the same as any other bus read or write cycle.
- Data channel transfers are treated as indivisable operations by the 80286. The Bus Unit will not execute a data or prefetch operation until a data channel word transfer between memory and the 80287 is complete.
- Data channel transfers to or from the 80287 are always word transfers. No single-byte transfers are required by the 80287, nor are they performed by the 80286.
- Word transfers to an odd-addressed memory location are performed as two byte-transfers. This results in three bus cycles (one word-transfer from the processor extension, and two byte-transfers to memory).

BUS UTILIZATION

Many features of the 80286 contribute to a high utilization of the iAPX 286 local bus. These features include:

- partitioning of CPU functions into individual processing units
- the Bus Unit guidelines for prefetch cycles, data cycles, and data channel transfers
- temporary Bus Unit address and data registers
- pipelined timing to allow back-to-back and overlapped bus operations

Depending on the particular program that is being executed and the speed of local memory, these architectural features can result in very high local bus utilization and bus efficiency. For typical types of software, about 60% to 70% of all bus operations will be instruction prefetches, and about 75%-85% of all bus clocks will be utilized by the CPU (assuming that memory responds with zero wait-states). If memories having one or more wait states are used, better than 90% of all available bus clocks will be used by the CPU. Long sequences of back-to-back bus operations are possible without the bus being idle, especially if a program involves string operations.

Designers can best take advantage of the high local bus throughput offered by pipelined timing and other features of the 80286 when interfacing the 80286 to local memory. The software tasks and data that are requested most often can reside in local memory where they can be quickly accessed. Less-used tasks and data can reside in remote memory on a public system bus, where access delays may be incurred due to bus arbitration and additional address and data buffers. Chapter Four discusses the performance tradeoffs of interfacing to fast vs. slow memories.

The job of implementing an iAPX 286 local bus around an 80286 CPU is made relatively simple by using several components specifically adapted to supporting the pipelined timing and status signals of the 80286. These interface components include:

- The 82284 Clock Generator, to generate the proper clock input for the 80286 CPU, to selectively enable and synchronize the 80286 $\overline{\text{READY}}$ input, and to handle the conditioning of the system RESET signal.
- The 82288 Bus Controller, to decode the 80286 status signals and to generate appropriate memory and I/O read/write commands, and data transceiver and address buffer control signals.
- 8282 octal latches, for latching local bus address lines or chip selects.
- 8286 data transceivers, for buffering the local bus data lines.

These devices support the pipelined timing of the 80286 local bus and combine functions that would take several dozen discrete components to perform. The remaining sections of this chapter describe how these circuits may be used to implement an iAPX 286 local bus.

GENERATING TIMING USING THE 82284 CLOCK GENERATOR

The 82284 Clock Generator provides the clock generation, $\overline{\text{READY}}$ timing, and RESET timing functions for the 80286 CPU and other iAPX 286 support devices. Figure 3-17 illustrates how the 82284 Clock Generator connects to an 80286 CPU.

In generating the clock signal for the system, the 82284 can use either a crystal or an external TTL-level signal as its frequency source. The CLK output signal is equal to the 80286 input source frequency and is double the specified processor clock frequency. The status lines $\overline{\text{S0}}$ and $\overline{\text{S1}}$ from the CPU are decoded by the 82284 to synchronize the 82284's PCLK (peripheral clock) output. This PCLK output is normally in phase with the 80286 internal processor clock.

The 82284 generates the $\overline{\text{READY}}$ input to the 80286, which the CPU uses either to insert wait states into bus operations or to terminate successful bus operations. This $\overline{\text{READY}}$ output is synchronized to the system clock, and can be selectively generated from either an asynchronous or a synchronous ready input to the 82284.

The third timing function of the 82284 Clock Generator is the generation of the 80286 RESET, or system RESET signal. The 82284 accepts an active-low $\overline{\text{RES}}$ input signal from a simple RC circuit or other Reset source, synchronizes it with the system CLK, and drives its RESET output to properly initialize the 80286 CPU and other system components.

The following sections describe each of these functions of the 82284 Clock Generator, and discuss some of the alternatives and considerations in designing an iAPX 286 system using the 82284.

Generating the System Clock

The 80286 requires a clock signal with fast rise and fall times (10 ns max) between low and high voltages of 0.6V low and 3.8V high. The maximum frequency of the clock input to the (8-MHz) 80286 is 16 MHz (20 MHz for a 10 MHz 80286-1). Since the 80286 internally uses dynamic cells, a minimum clock input frequency of 4 MHz is required to maintain the state of the CPU. Due to this minimum frequency requirement, the 80286 cannot be single-stepped by disabling the clock. The timing and voltage requirements for the CPU clock are shown in Figure 3-18.

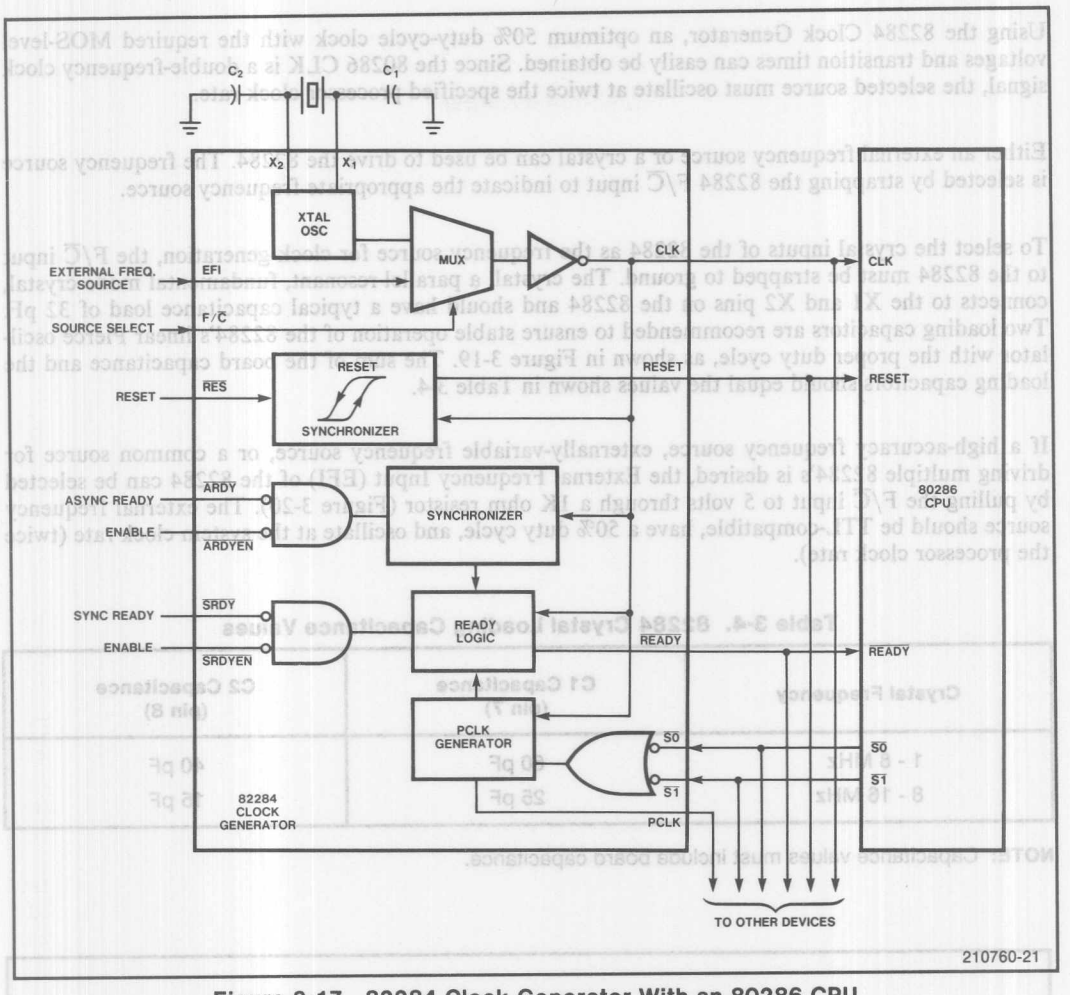


Figure 3-17. 82284 Clock Generator With an 80286 CPU

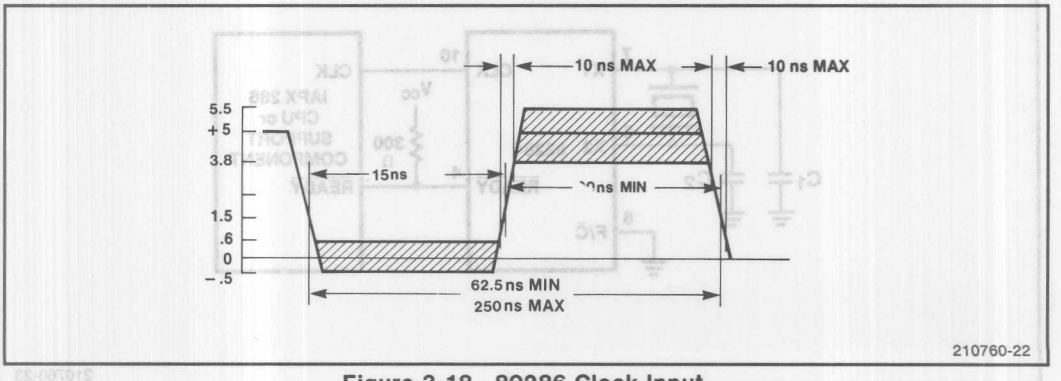


Figure 3-18. 80286 Clock Input

Using the 82284 Clock Generator, an optimum 50% duty-cycle clock with the required MOS-level voltages and transition times can easily be obtained. Since the 80286 CLK is a double-frequency clock signal, the selected source must oscillate at twice the specified processor clock rate.

Either an external frequency source or a crystal can be used to drive the 82284. The frequency source is selected by strapping the 82284 F/$\overline{C}$ input to indicate the appropriate frequency source.

To select the crystal inputs of the 82284 as the frequency source for clock generation, the F/$\overline{C}$ input to the 82284 must be strapped to ground. The crystal, a parallel-resonant, fundamental mode crystal, connects to the X1 and X2 pins on the 82284 and should have a typical capacitance load of 32 pF. Two loading capacitors are recommended to ensure stable operation of the 82284's linear Pierce oscillator with the proper duty cycle, as shown in Figure 3-19. The sum of the board capacitance and the loading capacitors should equal the values shown in Table 3-4.

If a high-accuracy frequency source, externally-variable frequency source, or a common source for driving multiple 82284's is desired, the External Frequency Input (EFI) of the 82284 can be selected by pulling the F/$\overline{C}$ input to 5 volts through a 1K ohm resistor (Figure 3-20). The external frequency source should be TTL-compatible, have a 50% duty cycle, and oscillate at the system clock rate (twice the processor clock rate).

Table 3-4. 82284 Crystal Loading Capacitance Values

Crystal Frequency	C1 Capacitance (pin 7)	C2 Capacitance (pin 8)
1 - 8 MHz	60 pF	40 pF
8 - 16 MHz	25 pF	15 pF

NOTE: Capacitance values must include board capacitance.

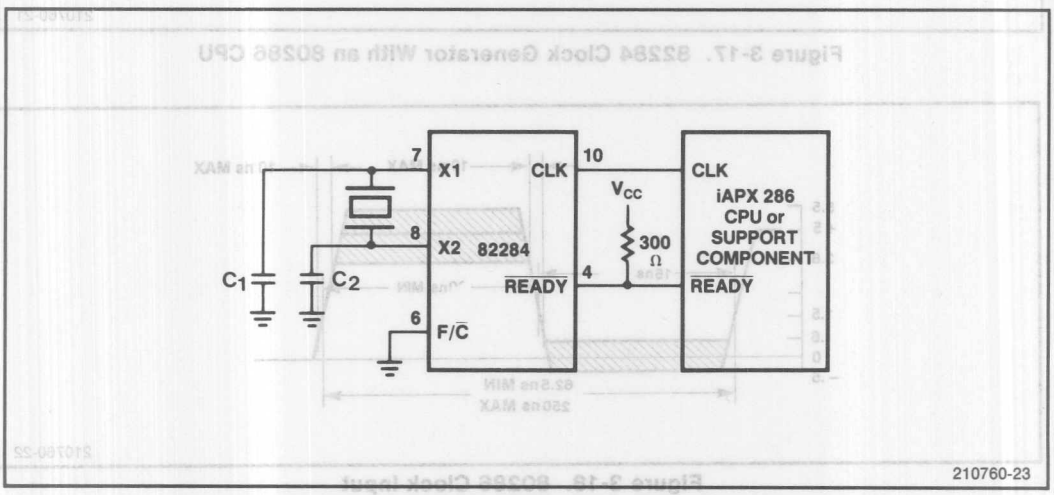


Figure 3-19. Recommended Crystal Connections to the 82284

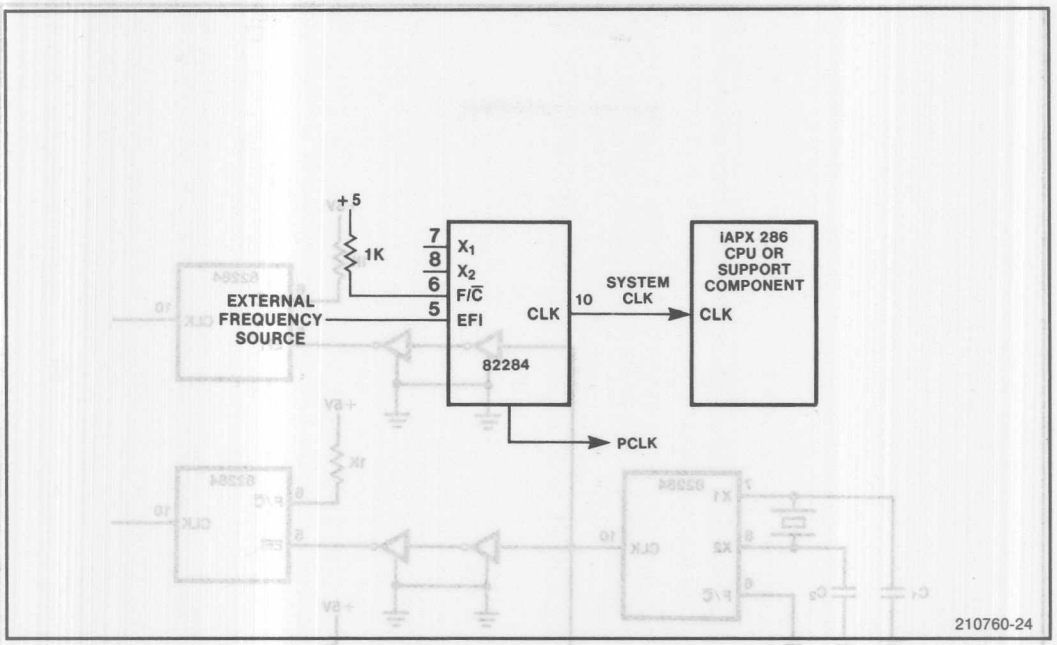


Figure 3-20. 82284 With External Frequency Source

If several sources of CLK are needed (in a multi-processor system, for example), multiple 82284 Clock Generators can be used, driven by a common frequency source. When multiple 82284's driven by a common source are distributed throughout a system, each 82284 should be driven by its own line from the source. To minimize noise in the system, each line should be a twisted pair driven by buffers like the 74LS04 with the ground of the twisted pair connecting the grounds of the source and receiver. To minimize clock skew, the lines to each 82284 should be of equal length.

A simple technique for generating a master frequency source for additional 82284's is shown in Figure 3-21. One 82284 with a crystal is used to generate the desired frequency. The CLK output from this 82284 drives the EFI input to the other 82284 Clock Generators in the system.

Since the CLK output is delayed from the EFI input by up to 30 ns, the CLK output of the master 82284 should not be used to drive both an 80286 and the EFI input of another 82284 (for a second 80286) if the two CPUs are to be synchronized. The variation on EFI-to-CLK delay over a range of 82284s may approach 10 to 15 ns. If, however, all 82284s are of the same package type, and have the same relative supply voltage, and operate in the same temperature environment, the variation will be reduced to between 5 and 10 ns.

When multiple processors share a common local bus, they should be driven with the same system CLK to optimize the transfer of bus control. A single 82284 Clock Generator can be used in this configuration, as shown in Figure 3-22. Each of the processors can share the common $\overline{\text{READY}}$ signal, since only one processor is permitted to use the bus at any given time. Processors that are not in control of the local bus will ignore any $\overline{\text{READY}}$ input.

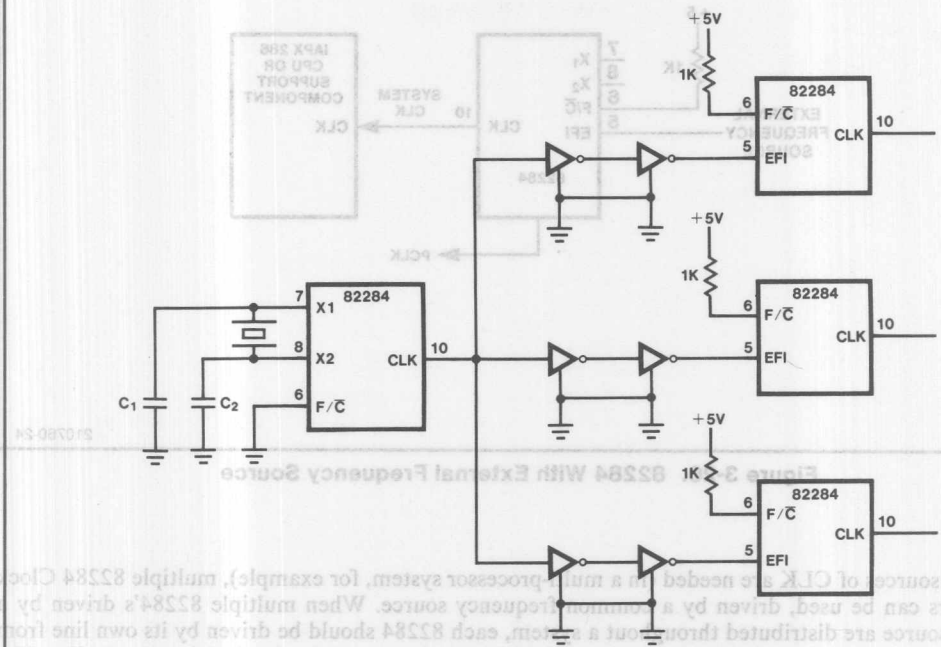


Figure 3-21. External Frequency for Multiple 82284s

The 82284 Clock Generator has two clock outputs: the system clock (CLK), which drives the CPU and support devices, and a peripheral clock (PCLK), which runs at one-half the frequency of the system CLK and is normally synchronized to the 80286 internal processor clock. CLK has a 50% duty cycle, matching its input, and so does PCLK. CLK has MOS-level drive characteristics, while PCLK is a TTL-level signal at half the frequency of the CLK output.

Figure 3-23 shows the relationship of CLK to PCLK. The maximum delay from CLK to PCLK is 40 ns. The 82284 synchronizes PCLK to the CPU internal processor clock at the start of the first bus operation following a RESET, as detected by $\overline{S1}$ or $\overline{S0}$ going low. Following this first bus cycle, PCLK will remain in phase with the internal processor clock.

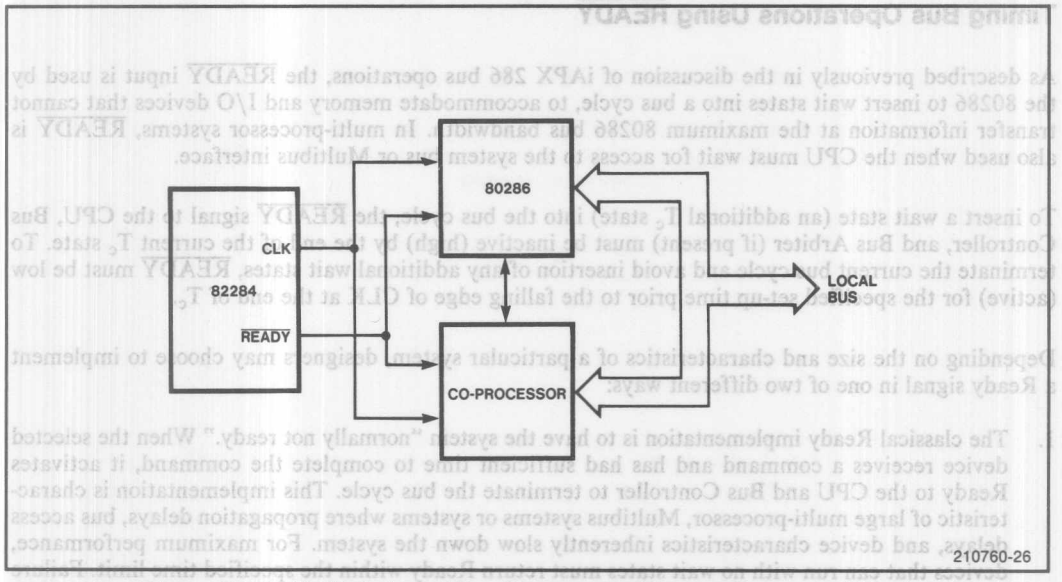


Figure 3-22. Multiple Local Bus Processors Share a Common 82284

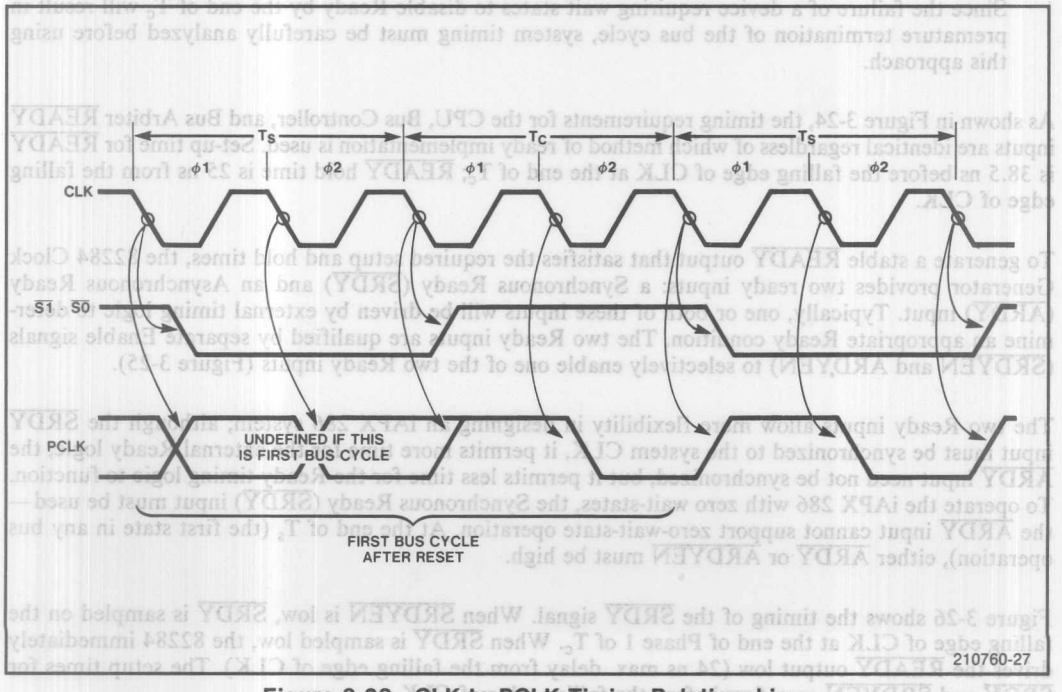


Figure 3-23. CLK to PCLK Timing Relationship

Timing Bus Operations Using $\overline{\text{READY}}$

As described previously in the discussion of iAPX 286 bus operations, the $\overline{\text{READY}}$ input is used by the 80286 to insert wait states into a bus cycle, to accommodate memory and I/O devices that cannot transfer information at the maximum 80286 bus bandwidth. In multi-processor systems, $\overline{\text{READY}}$ is also used when the CPU must wait for access to the system bus or Multibus interface.

To insert a wait state (an additional T_c state) into the bus cycle, the $\overline{\text{READY}}$ signal to the CPU, Bus Controller, and Bus Arbiter (if present) must be inactive (high) by the end of the current T_c state. To terminate the current bus cycle and avoid insertion of any additional wait states, $\overline{\text{READY}}$ must be low (active) for the specified set-up time prior to the falling edge of CLK at the end of T_c.

Depending on the size and characteristics of a particular system, designers may choose to implement a Ready signal in one of two different ways:

1. The classical Ready implementation is to have the system "normally not ready." When the selected device receives a command and has had sufficient time to complete the command, it activates Ready to the CPU and Bus Controller to terminate the bus cycle. This implementation is characteristic of large multi-processor, Multibus systems or systems where propagation delays, bus access delays, and device characteristics inherently slow down the system. For maximum performance, devices that can run with no wait states must return Ready within the specified time limit. Failure to respond in time will result in the insertion of one or more wait states.
2. An alternate technique is to have the system "normally ready." All devices are assumed to operate at the maximum CPU bus bandwidth. Devices that do not meet the requirement must disable Ready by the end of T_c to guarantee the insertion of wait states. This implementation is typically applied to small, single-CPU systems and reduces the logic required to control the $\overline{\text{READY}}$ signal. Since the failure of a device requiring wait states to disable Ready by the end of T_c will result in premature termination of the bus cycle, system timing must be carefully analyzed before using this approach.

As shown in Figure 3-24, the timing requirements for the CPU, Bus Controller, and Bus Arbiter $\overline{\text{READY}}$ inputs are identical regardless of which method of ready implementation is used. Set-up time for $\overline{\text{READY}}$ is 38.5 ns before the falling edge of CLK at the end of T_c; $\overline{\text{READY}}$ hold time is 25 ns from the falling edge of CLK.

To generate a stable $\overline{\text{READY}}$ output that satisfies the required setup and hold times, the 82284 Clock Generator provides two ready inputs: a Synchronous Ready ($\overline{\text{SRDY}}$) and an Asynchronous Ready ($\overline{\text{ARDY}}$) input. Typically, one or both of these inputs will be driven by external timing logic to determine an appropriate Ready condition. The two Ready inputs are qualified by separate Enable signals ($\overline{\text{SRDYEN}}$ and $\overline{\text{ARDYEN}}$) to selectively enable one of the two Ready inputs (Figure 3-25).

The two Ready inputs allow more flexibility in designing an iAPX 286 system; although the $\overline{\text{SRDY}}$ input must be synchronized to the system CLK, it permits more time for the external Ready logic; the $\overline{\text{ARDY}}$ input need not be synchronized, but it permits less time for the Ready-timing logic to function. To operate the iAPX 286 with zero wait-states, the Synchronous Ready ($\overline{\text{SRDY}}$) input must be used—the $\overline{\text{ARDY}}$ input cannot support zero-wait-state operation. At the end of T_s (the first state in any bus operation), either $\overline{\text{ARDY}}$ or $\overline{\text{ARDYEN}}$ must be high.

Figure 3-26 shows the timing of the $\overline{\text{SRDY}}$ signal. When $\overline{\text{SRDYEN}}$ is low, $\overline{\text{SRDY}}$ is sampled on the falling edge of CLK at the end of Phase 1 of T_c. When $\overline{\text{SRDY}}$ is sampled low, the 82284 immediately drives the $\overline{\text{READY}}$ output low (24 ns max. delay from the falling edge of CLK). The setup times for $\overline{\text{SRDY}}$ and $\overline{\text{SRDYEN}}$ are 15 ns before the falling edge of CLK.

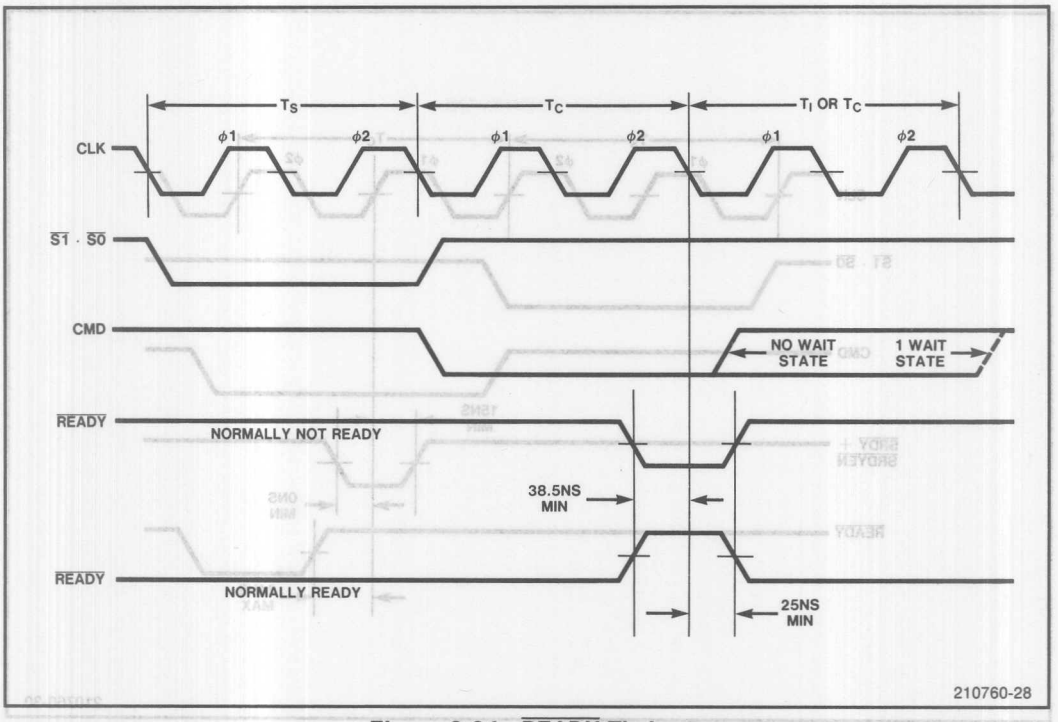


Figure 3-24. READY Timing

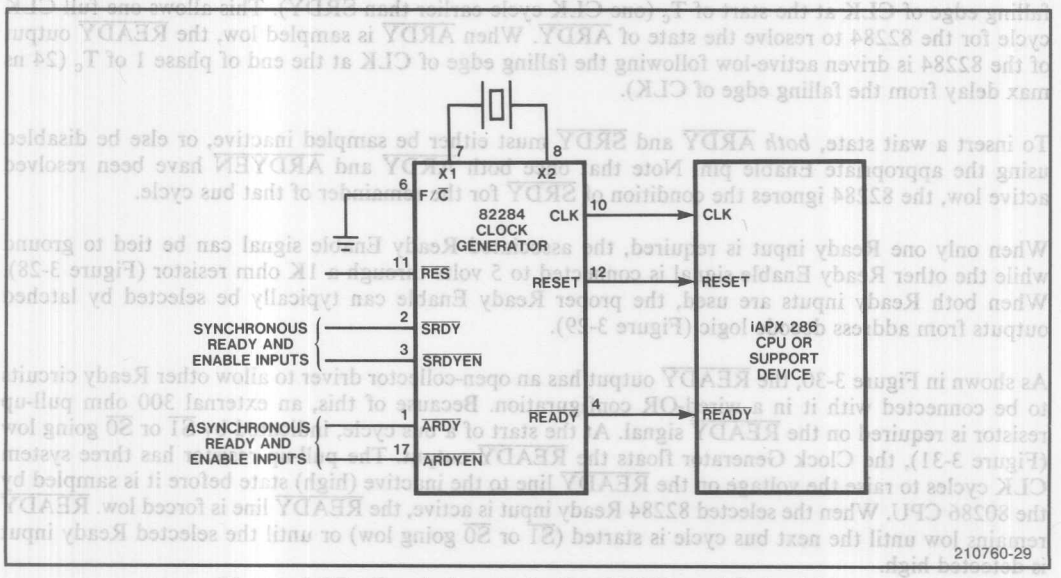


Figure 3-25. Ready Inputs to the 82284 and Output to the 80286 and 82288

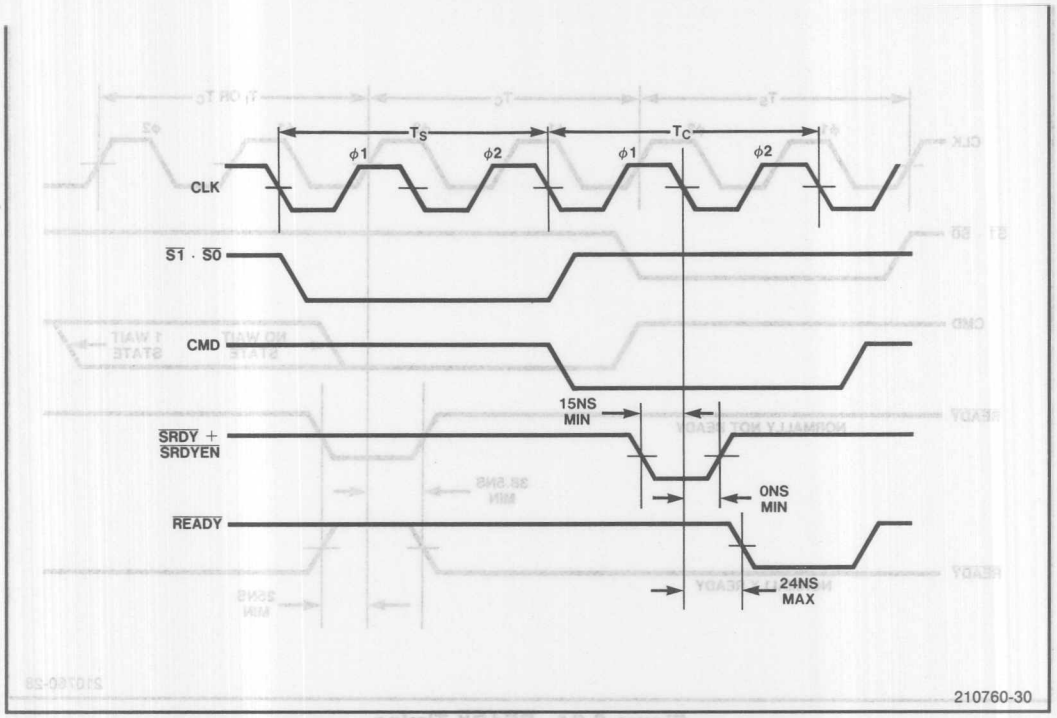


Figure 3-26. $\overline{\text{SRDY}}$ and $\overline{\text{READY}}$ Timing

Figure 3-27 shows the function of the $\overline{\text{ARDY}}$ input. When $\overline{\text{ARDYEN}}$ is low, $\overline{\text{ARDY}}$ is sampled on the falling edge of CLK at the start of T_c (one CLK cycle earlier than $\overline{\text{SRDY}}$). This allows one full CLK cycle for the 82284 to resolve the state of $\overline{\text{ARDY}}$. When $\overline{\text{ARDY}}$ is sampled low, the $\overline{\text{READY}}$ output of the 82284 is driven active-low following the falling edge of CLK at the end of phase 1 of T_c (24 ns max delay from the falling edge of CLK).

To insert a wait state, *both* $\overline{\text{ARDY}}$ and $\overline{\text{SRDY}}$ must either be sampled inactive, or else be disabled using the appropriate Enable pin. Note that once both $\overline{\text{ARDY}}$ and $\overline{\text{ARDYEN}}$ have been resolved active low, the 82284 ignores the condition of $\overline{\text{SRDY}}$ for the remainder of that bus cycle.

When only one Ready input is required, the associated Ready Enable signal can be tied to ground while the other Ready Enable signal is connected to 5 volts through a 1K ohm resistor (Figure 3-28). When both Ready inputs are used, the proper Ready Enable can typically be selected by latched outputs from address decode logic (Figure 3-29).

As shown in Figure 3-30, the $\overline{\text{READY}}$ output has an open-collector driver to allow other Ready circuits to be connected with it in a wired-OR configuration. Because of this, an external 300 ohm pull-up resistor is required on the $\overline{\text{READY}}$ signal. At the start of a bus cycle, indicated by $\overline{\text{S1}}$ or $\overline{\text{S0}}$ going low (Figure 3-31), the Clock Generator floats the $\overline{\text{READY}}$ output. The pull-up resistor has three system CLK cycles to raise the voltage on the $\overline{\text{READY}}$ line to the inactive (high) state before it is sampled by the 80286 CPU. When the selected 82284 Ready input is active, the $\overline{\text{READY}}$ line is forced low. $\overline{\text{READY}}$ remains low until the next bus cycle is started ($\overline{\text{S1}}$ or $\overline{\text{S0}}$ going low) or until the selected Ready input is detected high.

During RESET, the Clock Generator pulls $\overline{\text{READY}}$ low to force the Bus Controller into the idle state.

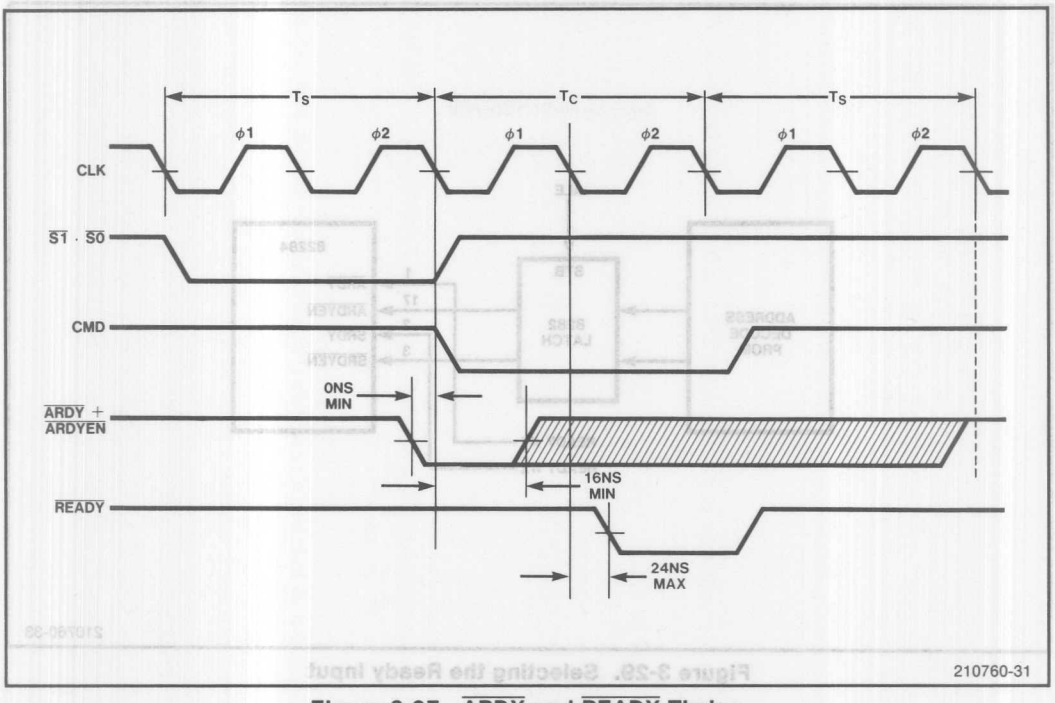


Figure 3-27. ARDY and READY Timing

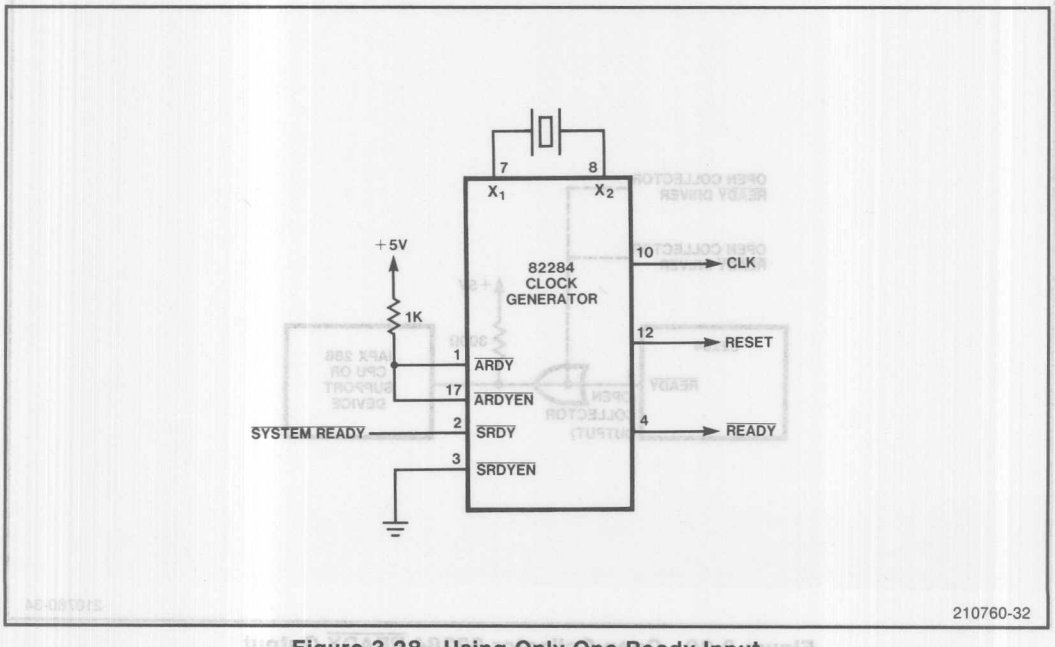


Figure 3-28. Using Only One Ready Input

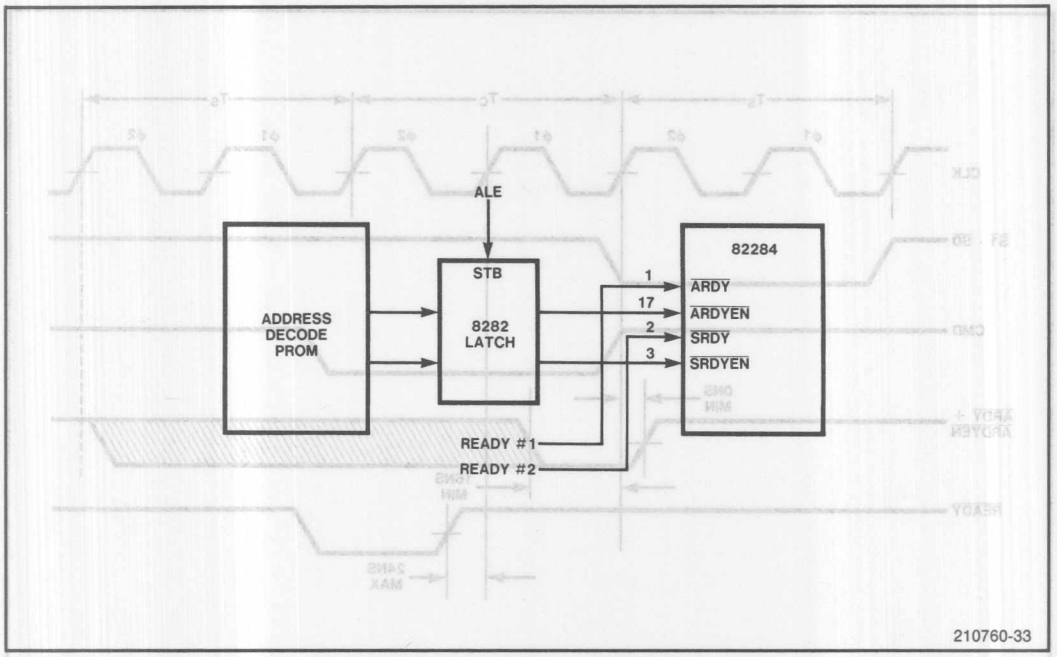


Figure 3-29. Selecting the Ready Input

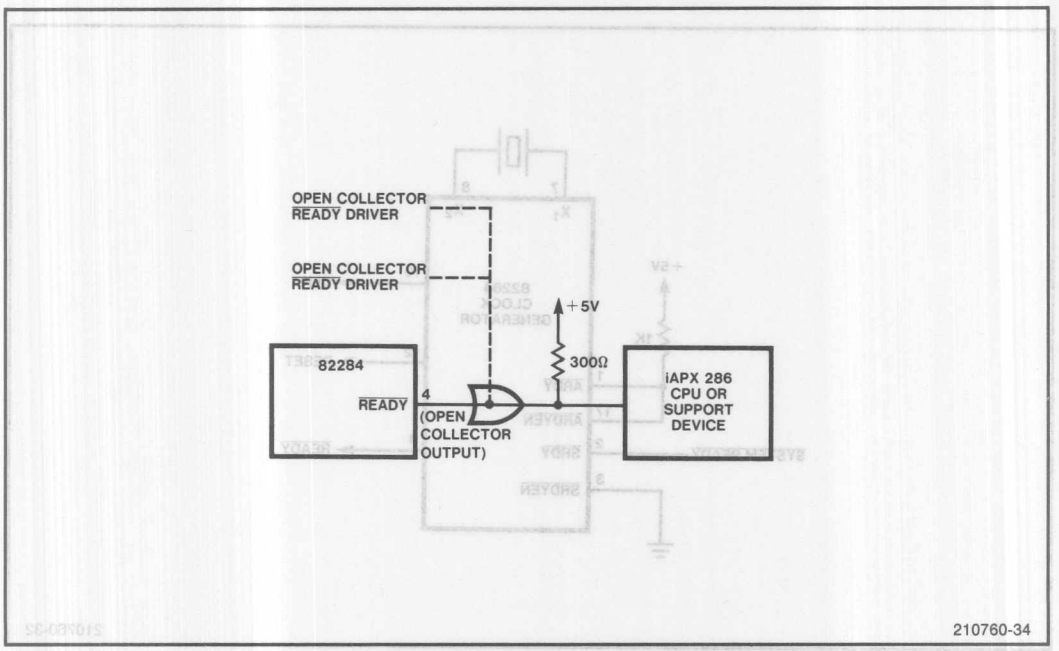


Figure 3-30. Open-Collector 82284 $\overline{\text{READY}}$ Output

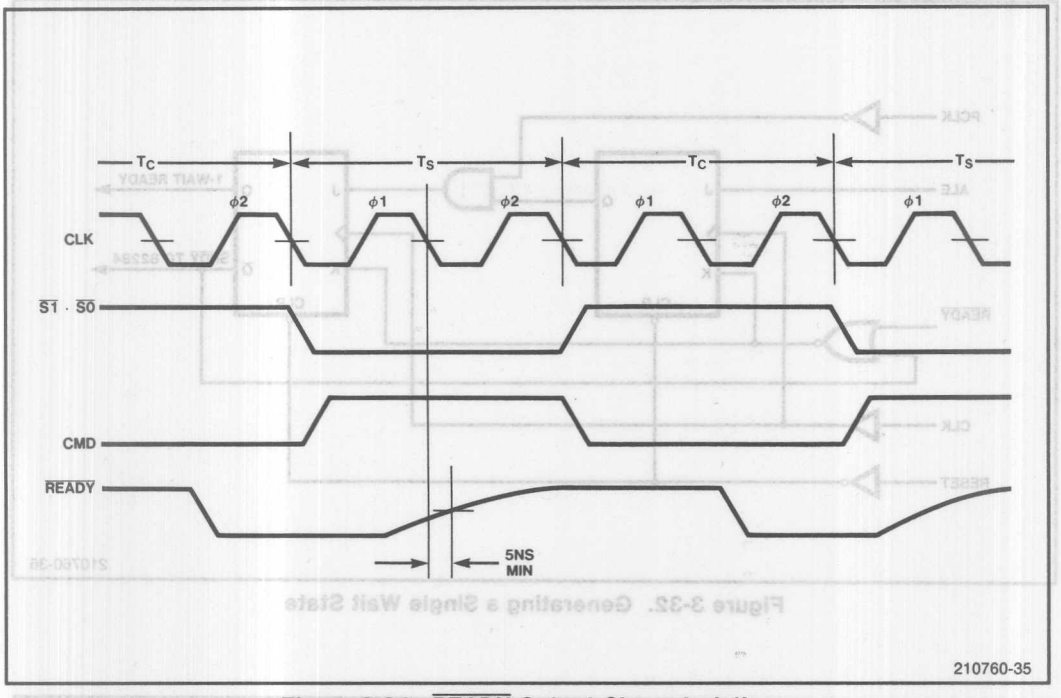


Figure 3-31. READY Output Characteristics

WAIT-STATE TIMING LOGIC

In the previous section, the operation of the SRDY and ARDY inputs to the Clock Generator were described. In this section, the wait-state timing logic that drives these Ready inputs are discussed, with specific examples for generating a fixed number of wait states.

With an 8-MHz 80286 CPU, fast memory and peripherals can typically operate with 0 or 1 wait states. In most cases, address-decode logic can be used to select the number of wait states required by a particular device. This address-decode logic is also used to generate the appropriate chip selects.

Figure 3-32 shows a simple method for generating one wait state in an iAPX 286 system. The timing for a typical bus cycle incorporating this single wait state is shown in Figure 3-33.

In the circuit shown, the output of the first flip-flop becomes active at the end of T_s. The output of the second flip-flop does not become activated until Phase 2 of the first T_c, after the 82284 has sampled SRDY high and caused the insertion of an additional wait state. During the following T_c state, the 82284 samples SRDY low and terminates the bus cycle by setting READY low. READY also resets the wait-state timing circuit in preparation for the following bus cycle.

For many existing memory and peripheral devices, one or more wait states will be required to operate with an 8-MHz 80286 CPU. Some peripheral devices may require more than two wait states. The circuit in Figure 3-34 shows a wait-state generator that inserts from 0 to 3 wait states into each bus cycle. This circuit can be extended to incorporate any number of wait states by adding an additional flip-flop for each wait state.

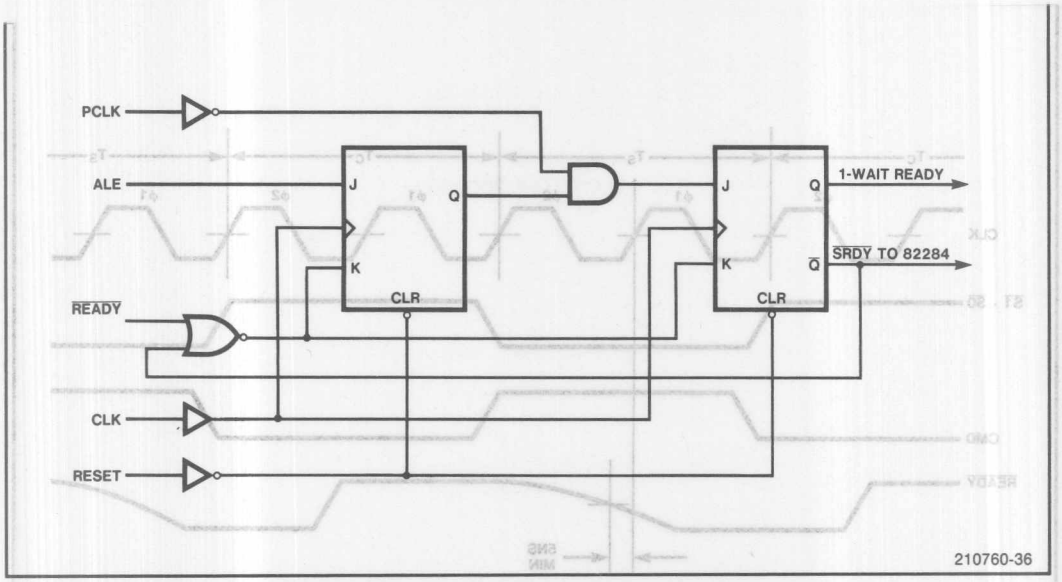


Figure 3-32. Generating a Single Wait State

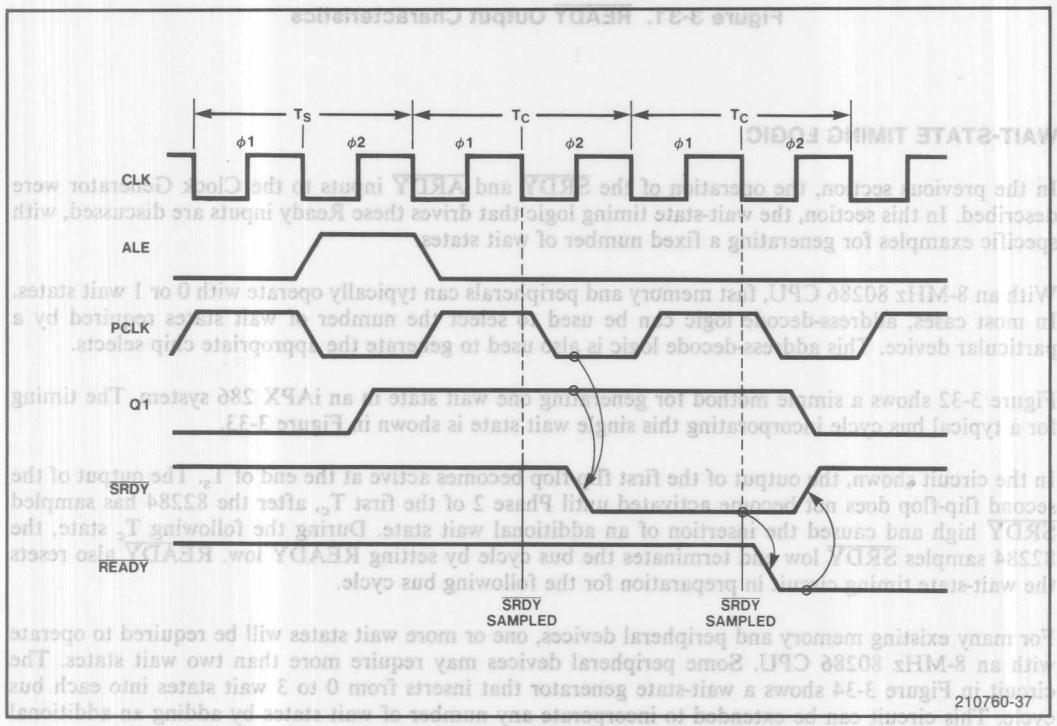


Figure 3-33. Timing for the Single Wait-State Generator

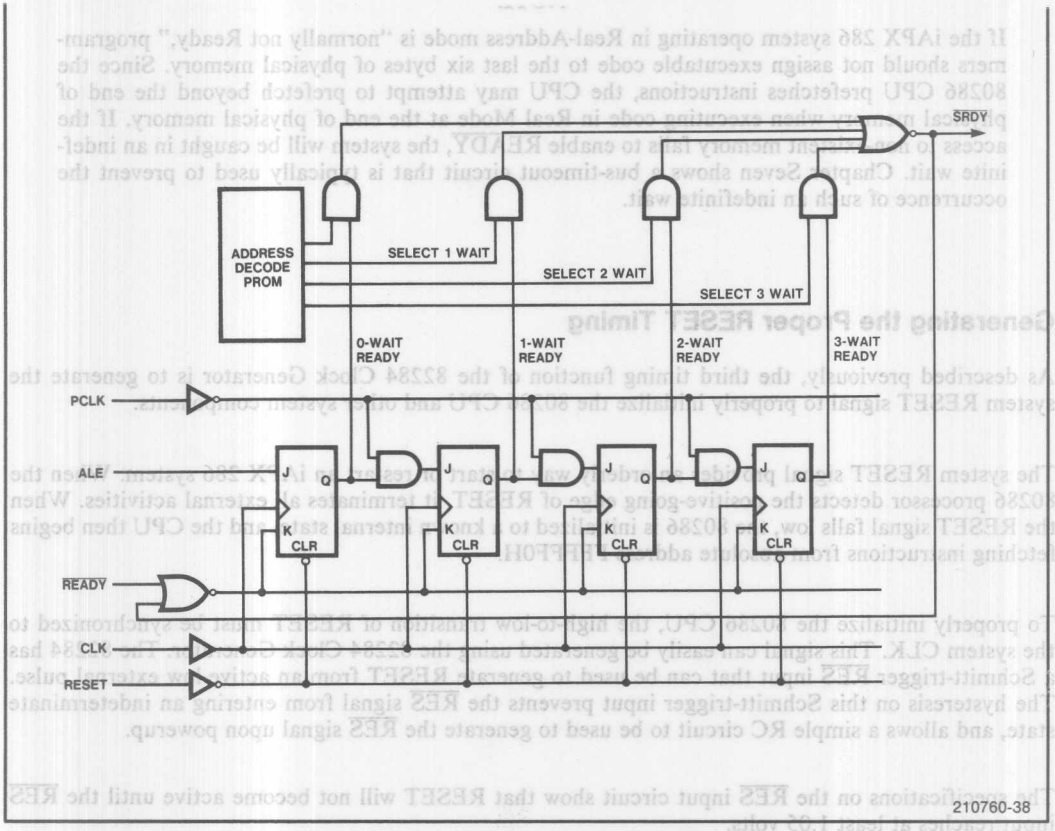


Figure 3-34. Generating from 0 to 3 Wait States

In the circuit shown, an address-decoder PROM selects the appropriate Ready signal to generate the proper number of wait states. The timing for this circuit is identical to that of the previous circuit, simply adding an additional wait state for each successive flip-flop.

These circuits are sufficient for generating $\overline{\text{SRDY}}$ timing for local devices when the required number of wait states is known. For device addresses that are not local, e.g. for devices that reside on a system bus or require other timing, the $\overline{\text{SRDYEN}}$ Enable signal to the 82284 should be driven inactive (high).

Memory and I/O devices configured on the Multibus interface generate their own Ready timing through the Multibus $\overline{\text{XACK}}$ signal (shown in Figure 3-41 in the next section). $\overline{\text{XACK}}$ can be used to drive the $\overline{\text{ARDY}}$ input to the 82284, while an address-decoder recognizes addresses mapped onto the Multibus and selects the $\overline{\text{ARDYEN}}$ Enable line. The 82284 will automatically insert wait states into the bus cycle until $\overline{\text{XACK}}$ becomes active low. More information on designing a Multibus interface for the iAPX 286 is given in Chapter Seven.

NOTE

If the iAPX 286 system operating in Real-Address mode is "normally not Ready," programmers should not assign executable code to the last six bytes of physical memory. Since the 80286 CPU prefetches instructions, the CPU may attempt to prefetch beyond the end of physical memory when executing code in Real Mode at the end of physical memory. If the access to non-existent memory fails to enable $\overline{\text{READY}}$, the system will be caught in an indefinite wait. Chapter Seven shows a bus-timeout circuit that is typically used to prevent the occurrence of such an indefinite wait.

Generating the Proper RESET Timing

As described previously, the third timing function of the 82284 Clock Generator is to generate the system RESET signal to properly initialize the 80286 CPU and other system components.

The system RESET signal provides an orderly way to start or restart an iAPX 286 system. When the 80286 processor detects the positive-going edge of RESET, it terminates all external activities. When the RESET signal falls low, the 80286 is initialized to a known internal state, and the CPU then begins fetching instructions from absolute address FFFFF0H.

To properly initialize the 80286 CPU, the high-to-low transition of RESET must be synchronized to the system CLK. This signal can easily be generated using the 82284 Clock Generator. The 82284 has a Schmitt-trigger $\overline{\text{RES}}$ input that can be used to generate RESET from an active-low external pulse. The hysteresis on this Schmitt-trigger input prevents the $\overline{\text{RES}}$ signal from entering an indeterminate state, and allows a simple RC circuit to be used to generate the $\overline{\text{RES}}$ signal upon powerup.

The specifications on the $\overline{\text{RES}}$ input circuit show that RESET will not become active until the $\overline{\text{RES}}$ input reaches at least 1.05 volts.

To guarantee RESET upon power-up, the $\overline{\text{RES}}$ input must remain below 1.05 volts for 5 milliseconds after Vcc reaches the minimum supply voltage of 4.5 volts. Following this event, $\overline{\text{RES}}$ must still remain below 1.05 volts for 16 processor clock cycles to allow for initializing the CPU. Figure 3-35 shows a simple RC circuit that will keep $\overline{\text{RES}}$ low long enough to satisfy both requirements.

The 82284 RESET output meets all of the requirements for the 80286 CPU RESET input. The RESET output is also available as a system RESET to other devices in the system, as shown in Figure 3-36.

During RESET, the 80286 internal processor clock is initialized (the divide by two counter is reset). Following the rising edge of RESET, the next CLK cycle will start Phase 2 of a processor clock cycle (see Figure 3-37). The 82284 synchronizes the 82284 PCLK output to the 80286 internal processor clock at the start of the first bus cycle after RESET.

Since the 82284 synchronizes RESET to the system CLK, a delay of one or two clock cycles is introduced before the RESET output is driven high. The 82284 will attempt to synchronize the rising edge of RESET if Vcc and CLK are correct at that time (Figure 3-38). However, since the rising edge of RESET may occur during power-up, when Vcc and CLK are not valid, the rising edge of RESET may be asynchronous to the CLK. A synchronized rising-edge of RESET is necessary only for clock-synchronous, multiprocessor systems.

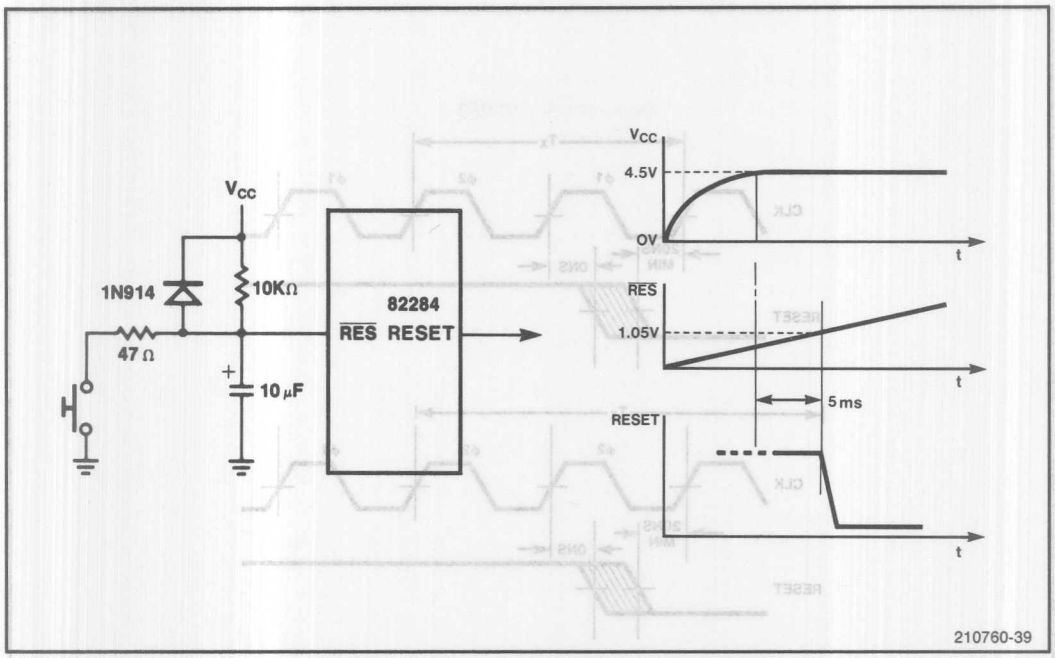


Figure 3-35. Typical RC RESET Timing Circuit

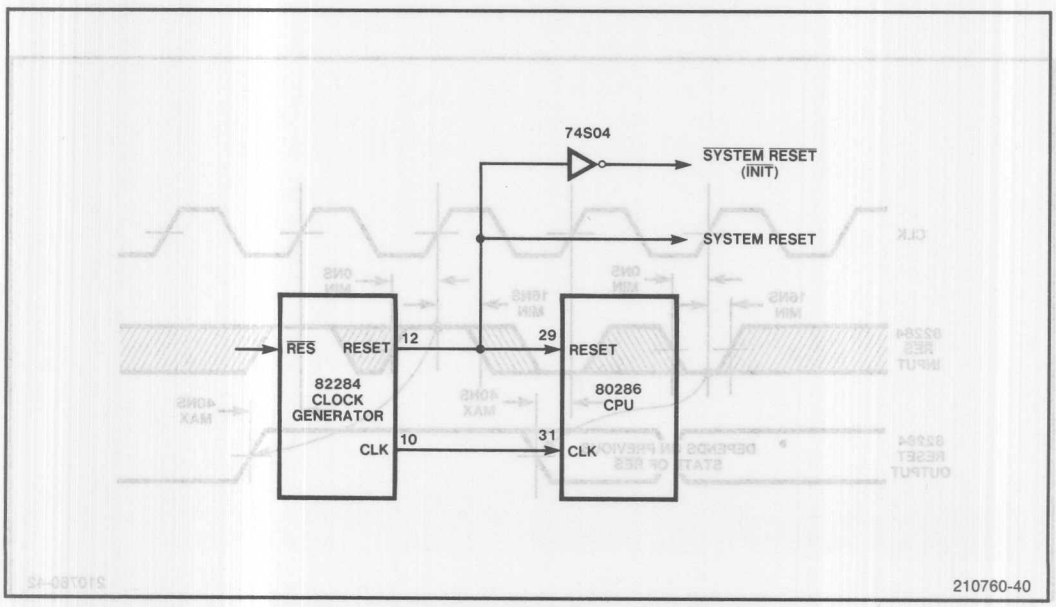


Figure 3-36. 80286 Reset and System Reset

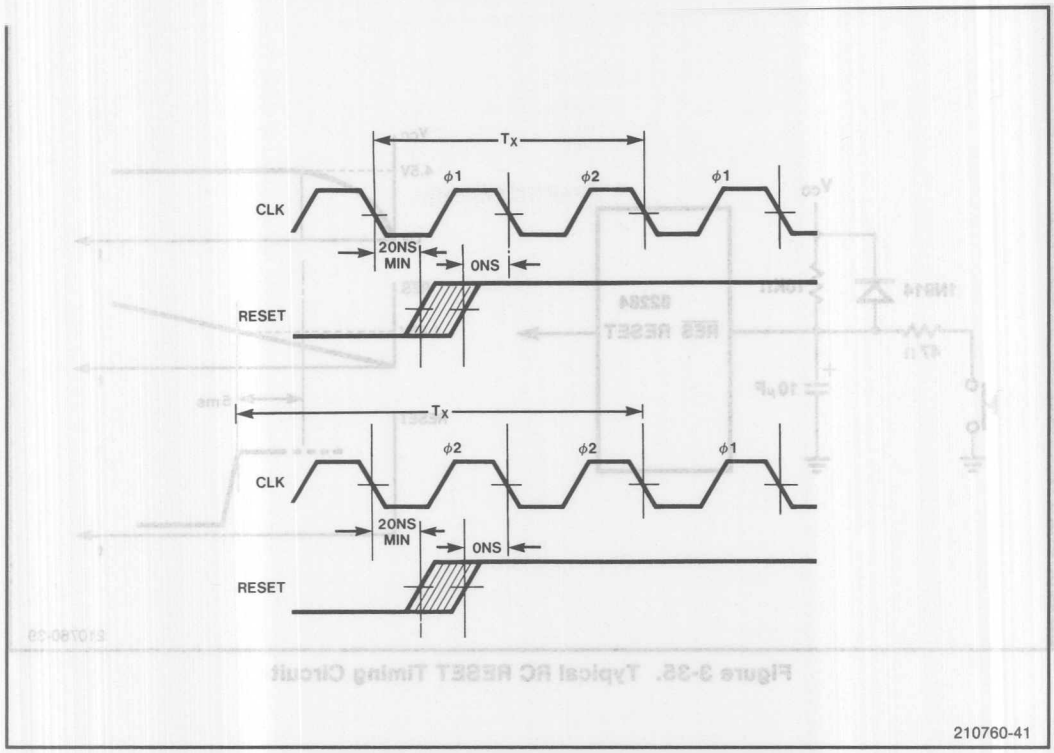


Figure 3-37. Processor Clock Synchronization

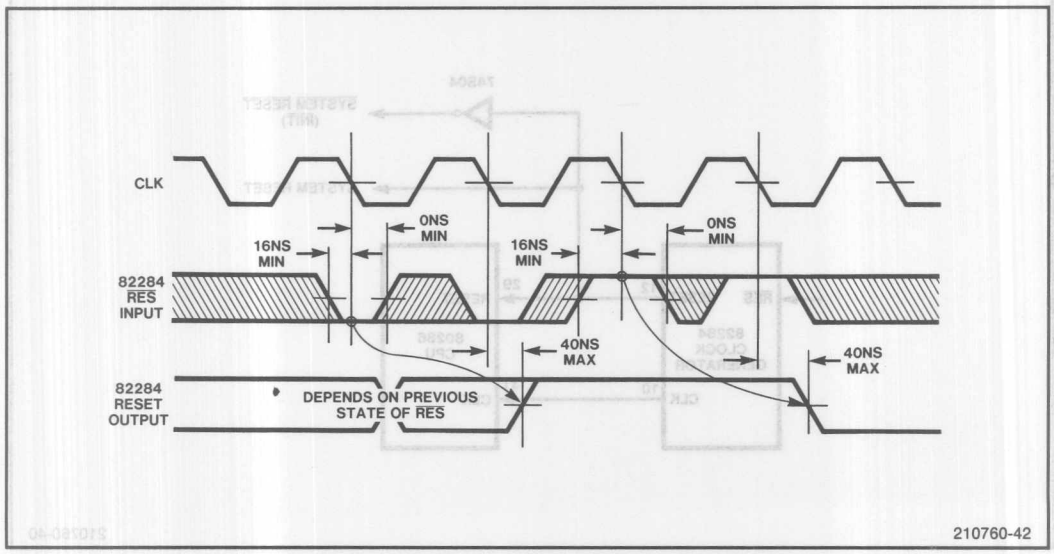


Figure 3-38. RESET Synchronized to CLK

Synchronizing Processor Clocks in a Multi-Processor System

In a multi-processor system where each processor must be clock-synchronous, all of the processor clocks must be synchronized following the initial power-up RESET. Since the CPU does not normally see the rising edge of a RESET signal during power-up due to delays between the time voltage is applied and the time that the processor is functioning properly, a second RESET pulse must be generated.

This second RESET pulse cannot be provided directly by the 82284 Clock Generator; additional logic is required. Figure 3-39 shows a circuit that provides a fully-synchronous RESET signal by generating a second RESET pulse that is triggered by the completion of the power-on reset. Timing for the circuit is shown in figure 3-40.

The circuit actually provides two individual reset signals: CPU RESET, a synchronous reset signal that drives the 80286 CPU's in the system, and SYSTEM RESET, providing only a single RESET pulse for peripheral devices that need not be synchronized to the processor clock. The circuit also generates PCLK for the system (the 82284 PCLK signal should be ignored).

In operation, the normal power-on RESET occurs with the rising edge of CPU RESET not being seen by the processors. When the RESET input goes low, the circuit automatically initiates a second RESET that is synchronous to the system clock. The CPU RESET output goes low for one CLK cycle and then goes high for seventeen cycles (the 80286 requires a minimum RESET pulse width of sixteen CLK cycles subsequent to power-up reset). Both edges of this second reset pulse are synchronous to

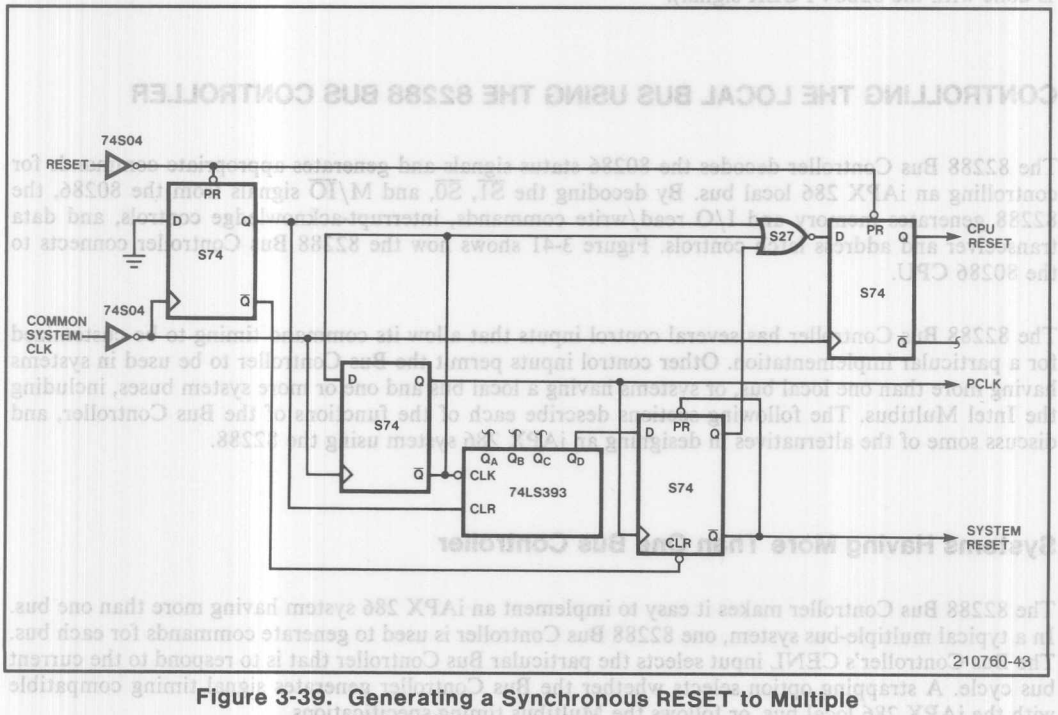


Figure 3-39. Generating a Synchronous RESET to Multiple 80286 CPUs

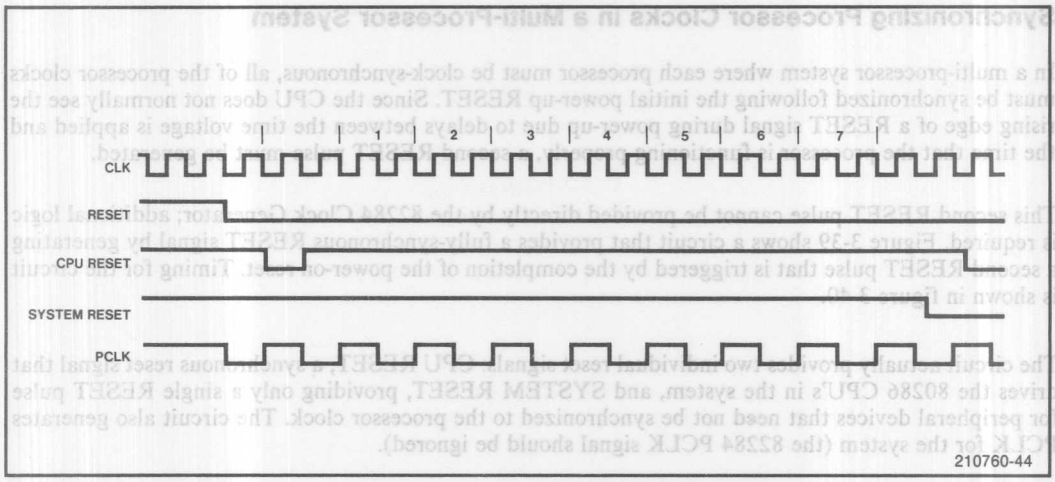


Figure 3-40. Synchronous Reset Circuit Timing

CLK and to PCLK (the pulse is derived from the PCLK signal). SYSTEM RESET remains high from power-up until the CLK cycle before the falling edge of the second CPU RESET pulse. Since the synchronous reset guarantees that PCLK is in phase with the processor clock, it is not necessary to synchronize PCLK to the processor clock via $\overline{S1}$ or $\overline{S0}$ going low at the start of the first bus cycle (as is done with the 82284 PCLK signal).

CONTROLLING THE LOCAL BUS USING THE 82288 BUS CONTROLLER

The 82288 Bus Controller decodes the 80286 status signals and generates appropriate commands for controlling an iAPX 286 local bus. By decoding the $\overline{S1}$, $\overline{S0}$, and M/$\overline{IO}$ signals from the 80286, the 82288 generates memory and I/O read/write commands, interrupt-acknowledge controls, and data transceiver and address latch controls. Figure 3-41 shows how the 82288 Bus Controller connects to the 80286 CPU.

The 82288 Bus Controller has several control inputs that allow its command timing to be customized for a particular implementation. Other control inputs permit the Bus Controller to be used in systems having more than one local bus, or systems having a local bus and one or more system buses, including the Intel Multibus. The following sections describe each of the functions of the Bus Controller, and discuss some of the alternatives in designing an iAPX 286 system using the 82288.

Systems Having More Than One Bus Controller

The 82288 Bus Controller makes it easy to implement an iAPX 286 system having more than one bus. In a typical multiple-bus system, one 82288 Bus Controller is used to generate commands for each bus. The Bus Controller's CENL input selects the particular Bus Controller that is to respond to the current bus cycle. A strapping option selects whether the Bus Controller generates signal timing compatible with the iAPX 286 local bus, or follows the Multibus timing specifications.

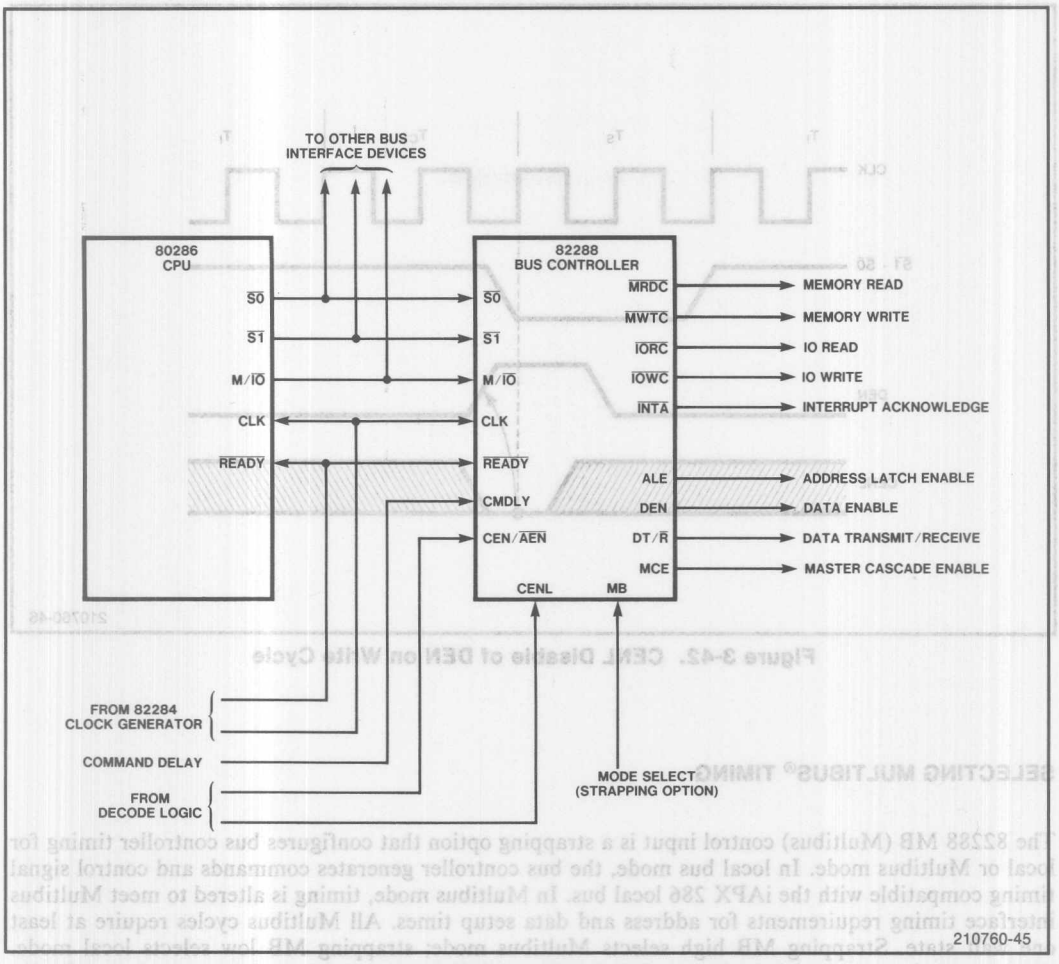


Figure 3-41. 82288 Bus Controller with an 80286 CPU

SELECTING A CONTROLLER

The 82288 CENL (Command Enable Latched) input selects whether the Bus Controller will respond to the current bus cycle. This input is sampled at the start of the first T_c state of each bus cycle. For systems that use only one Bus Controller, this input may be strapped high.

In systems that use multiple Controllers, address-decode logic typically selects the Bus Controller that executes the current bus cycle. CENL set-up time is 20 ns; sampling occurs on the falling edge of CLK between T_s and T_c. If CENL is sampled high, the Controller ignores the current bus cycle (e.g., DEN and commands are not asserted) and waits for the next cycle to begin. It should be noted that in the case of a write cycle, DEN is already active when CENL is sampled. Therefore, when CENL is sampled low during a write, DEN will be driven low to disable the data buffers within 35 ns, as shown in Figure 3-42.

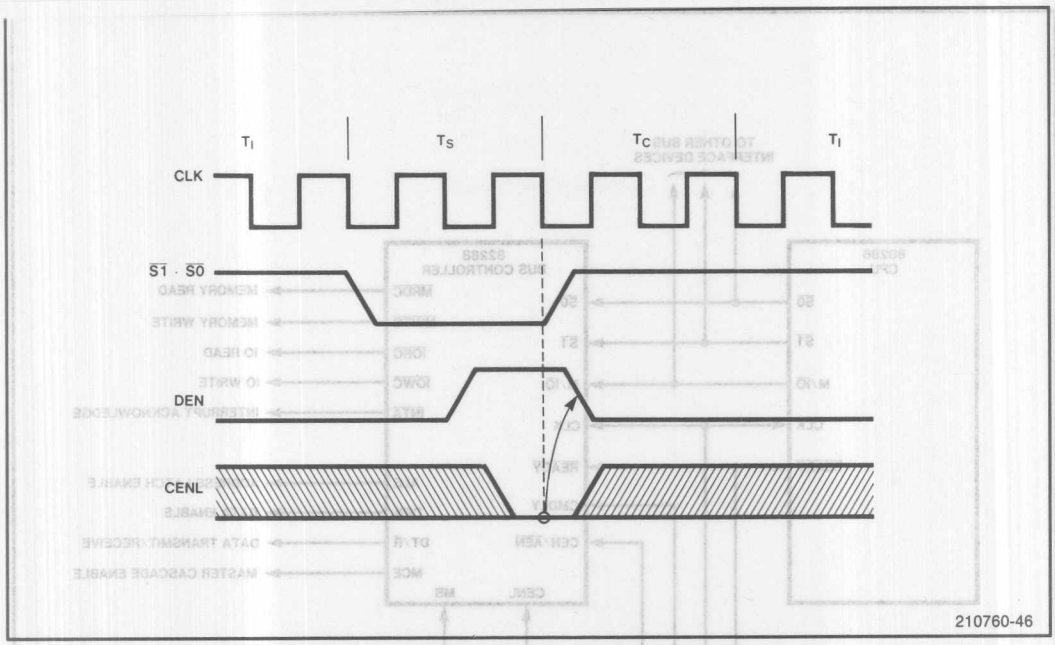


Figure 3-42. CENL Disable of DEN on Write Cycle

SELECTING MULTIBUS® TIMING

The 82288 MB (Multibus) control input is a strapping option that configures bus controller timing for local or Multibus mode. In local bus mode, the bus controller generates commands and control signal timing compatible with the iAPX 286 local bus. In Multibus mode, timing is altered to meet Multibus interface timing requirements for address and data setup times. All Multibus cycles require at least one wait state. Strapping MB high selects Multibus mode; strapping MB low selects local mode. Figure 3-43 shows typical bus command timing for an 82288 operating in both local mode and Multibus mode. Chapter Seven provides detailed guidelines for implementing a Multibus interface for the iAPX 286.

Figure 3-44 shows two 82288 bus controllers being used in an iAPX system. One controller interfaces the 80286 CPU to a local bus and is configured for local bus mode (MB strapped to ground). The second controller is configured in Multibus mode (MB strapped to +5V) and connects the CPU to the Multibus interface. The local bus controller also has CEN strapped to +5V. CMDLY is tied low; commands are not delayed. Address decode logic selects one of the two bus controllers via the CENL inputs. The select signal for the Multibus controller also enables the 82289 Bus Arbiter.

Access to a multimaster system bus such as the Multibus is controlled by the 82289 Bus Arbiter. The Bus Arbiter decodes the $\overline{S1}$, $\overline{S0}$, M/$\overline{IO}$, and SYSB/$\overline{RESB}$ lines to determine when the 80286 requires access to the multimaster bus. After requesting and gaining control of the bus, the 82289 enables the Bus Controller and address latches to allow the access to take place. Chapter Seven describes the operation and use of the 82289 Bus Arbiter in detail.

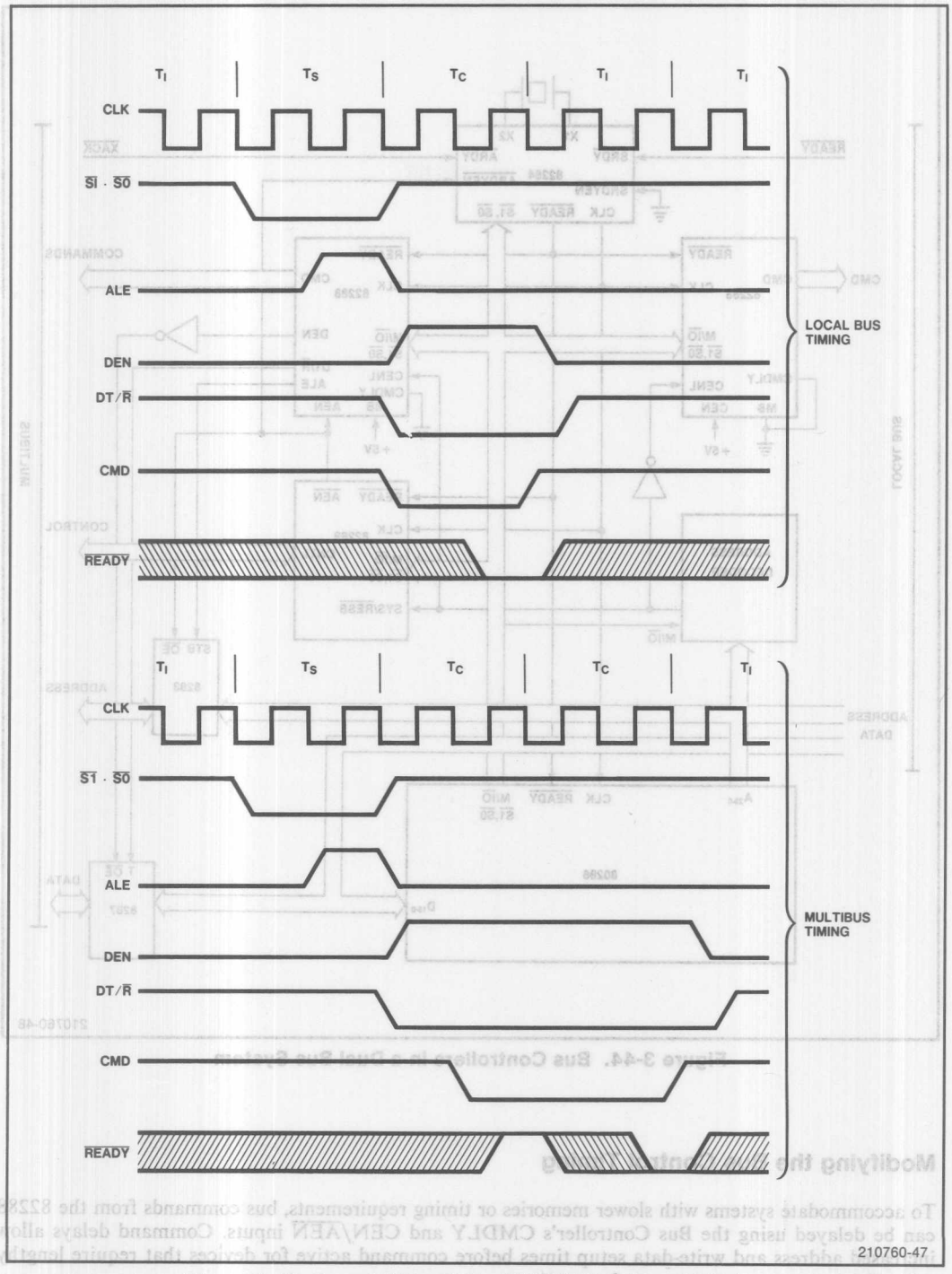


Figure 3-43. Local (MB=0) Vs. MULTIBUS® (B=1) Timing

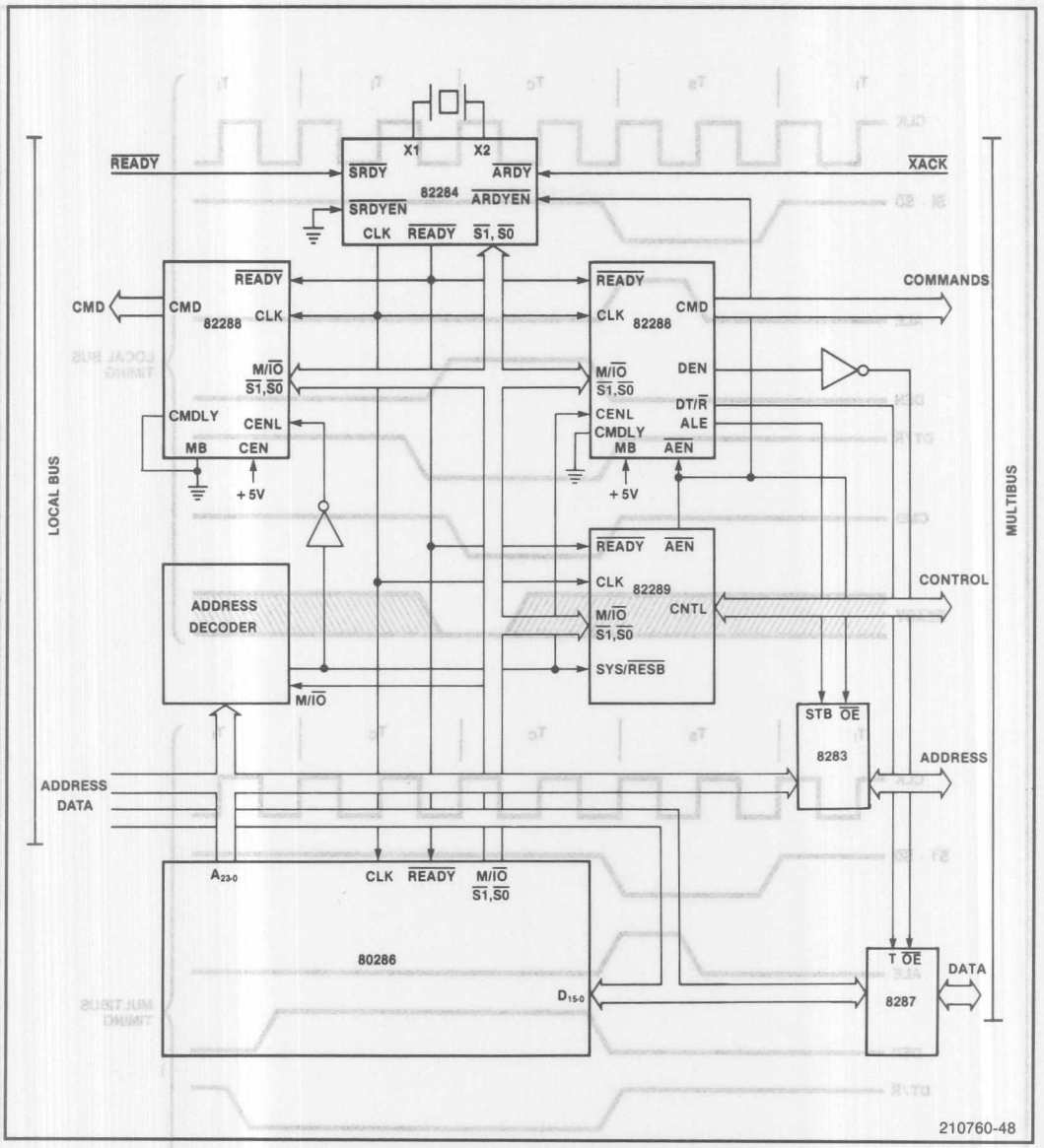


Figure 3-44. Bus Controllers in a Dual Bus System

Modifying the Bus Control Timing

To accommodate systems with slower memories or timing requirements, bus commands from the 82288 can be delayed using the Bus Controller's CMDLY and CEN/$\overline{\text{AEN}}$ inputs. Command delays allow increased address and write-data setup times before command active for devices that require lengthy setup times.

CMDLY (COMMAND DELAY)

The 82288 CMDLY input delays command generation by one or more CLK cycles. Each CLK delay allows at least 62.5 ns additional set-up time from address/chip select to command active. Command delays may be required, for example, on write operations when write data must be valid before the $\overline{WR}$ command is asserted. When no delays are required, CMDLY can be strapped to ground to always issue the command as soon as possible.

A simple method for generating 0 or 1 command delays is shown in Figure 3-45. The CMDLY input to the Bus Controller is driven by the output of an address-decode PROM gated to the Bus Controller's Address Latch Enable (ALE). CMDLY is driven high while ALE is active, but only when the PROM output is also high. CMDLY falls low after ALE is driven low. To satisfy 82288 timing requirements, the sum of PROM access time plus gate delay must not exceed 107 ns.

For devices that require more than one command delay, a counter can be used to generate the appropriate number of delays (Figure 3-46). Set-up time for CMDLY is 20 ns before the falling edge of CLK. Note that delaying commands does not lengthen the bus cycle, nor does CMDLY affect the DEN signal controlling the data transceivers. For example, if $\overline{READY}$ is sampled active low before CMDLY is sampled high, the bus cycle will be terminated, with no command being issued (see Figure 3-47).

When the Bus Controller is configured in Multibus mode, commands are automatically delayed for the proper duration (CMDLY should be strapped low).

CEN/$\overline{AEN}$ (Command Enable/Address Enable)

The 82288 CEN/$\overline{AEN}$ input is a dual-function, asynchronous input that disables commands and control signals. The function of the input depends on the mode of the Bus Controller. In local mode (MB=0), the input functions as CEN; in Multibus mode (MB=1), the input functions as AEN.

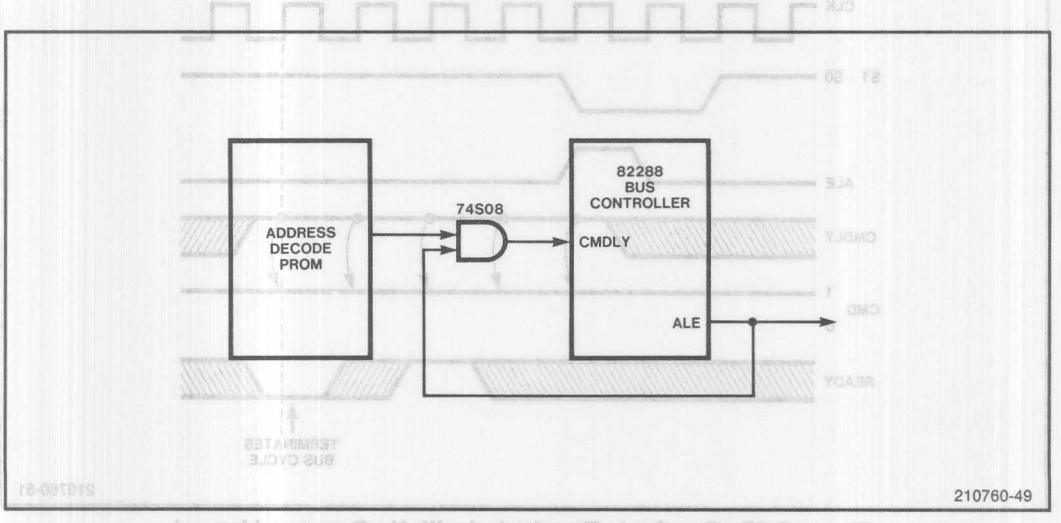
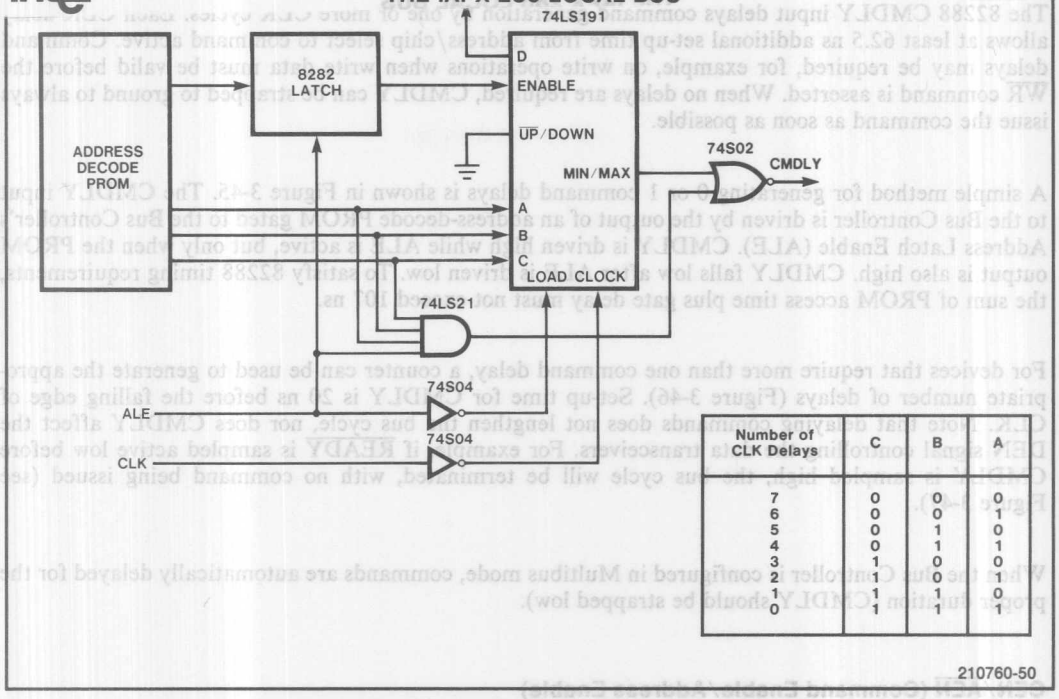


Figure 3-45. Generating CMDLY For 0 or 1 CLK Delays



Number of CLK Delays	C	B	A
7	0	0	0
6	0	0	1
5	0	1	0
4	0	1	1
3	1	0	0
2	1	0	1
1	1	1	0
0	1	1	1

Figure 3-46. Generating 0 to 7 Command Delays

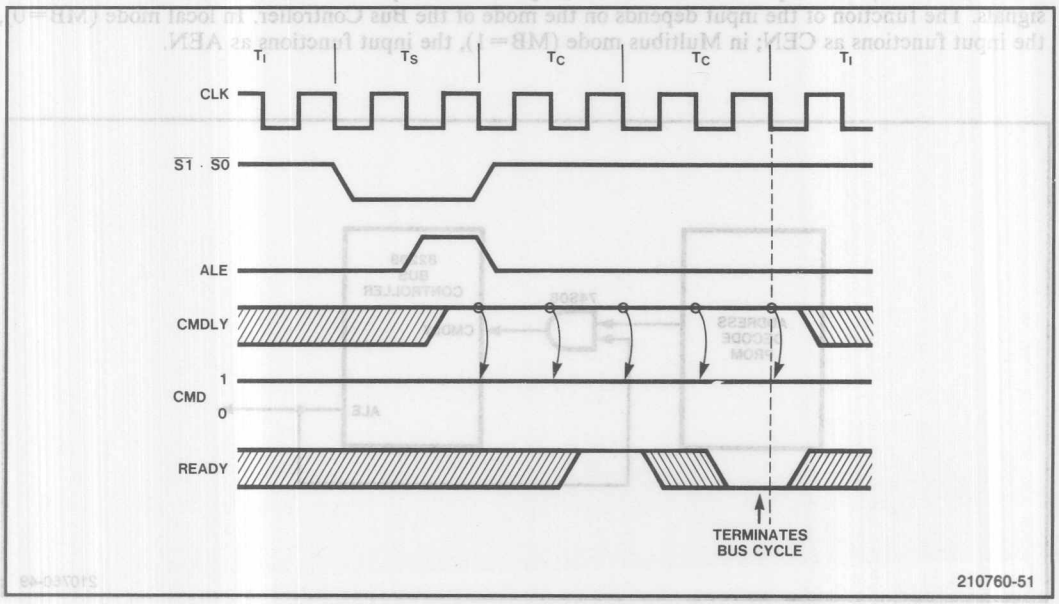


Figure 3-47. Bus Cycle Terminated with No Command Issued

CEN provides the Bus Controller with an additional command enable signal that can be used for address decoding or delaying commands and control signals asynchronously to the system clock. In contrast to CENL, CEN is not latched internally and, when used as part of an address decoding scheme, requires an external latch.

CEN contrasts with CMDLY in its ability to delay both command signals and DEN asynchronously to the system clock. Whereas the minimum CMDLY controlled delay is 62.5 ns (one system clock), no minimum is required for CEN.

As shown in Figure 3-48, when CEN is low, commands and DEN are inactive. When CEN goes high, commands and control signals immediately go active (up to 25 ns max delay).

$\overline{AEN}$ is a signal typically generated by the 82289 Bus Arbiter to control commands and data during accesses to the Multibus interface. For Multibus bus cycles, address decode logic enables the 82289 Bus Arbiter (SYSB/$\overline{RESB}$) and 82288 Bus Controller (CENL) assigned to the Multibus interface. After requesting and gaining control of the interface, the 82289 Bus Arbiter outputs the active-low $\overline{AEN}$ signal to the Bus Controller to enable the Multibus commands and data transceivers.

$\overline{AEN}$ is asynchronous to the system CLK. When $\overline{AEN}$ goes high, the Bus Controller drives DEN inactive and its command lines enter the tri-state OFF condition. Chapter Seven describes the use of $\overline{AEN}$ in greater detail.

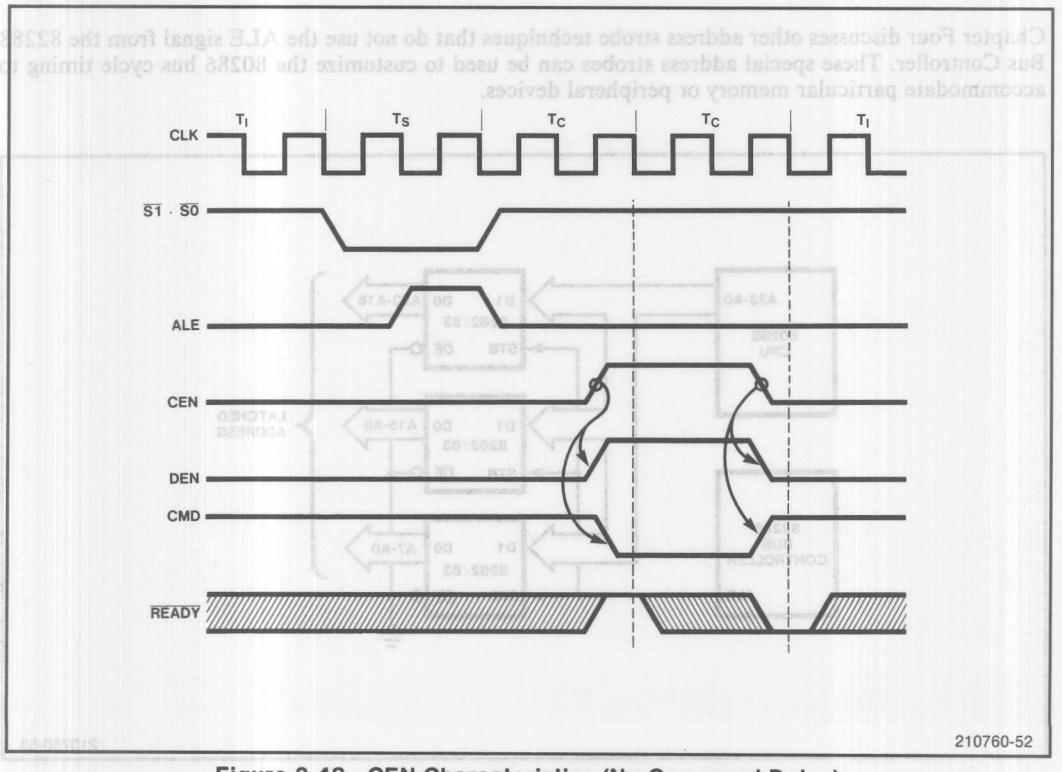


Figure 3-48. CEN Characteristics (No Command Delay)

Another operating mode of the 82288 allows tri-stating the Bus Controller's command outputs in non-Multibus mode. To do this, the CEN/$\overline{\text{AEN}}$ input is tied HIGH and the MB input is used to control the command outputs. MB must be stable 20 ns before the falling edge of CLK and remain so for 5 ns after CLK. Changing MB to HIGH will tri-state the command outputs on the falling edge of CLK. Changing MB to LOW will enable the control outputs first, then the command outputs will go active two CLK cycles later if a command is active.

LOCAL BUS DESIGN CONSIDERATIONS

Address Bus Interface

iAPX 286 systems operate with a 24-bit address bus, using the pipelined address timing described previously. Since the majority of system memories and peripherals require stable address and chip-select inputs for the duration of the bus cycle, the address bus and/or decoded chip-selects should be latched.

The 82288 Bus Controller provides an ALE signal to capture the address in 8282 (non-inverting) or 8283 (inverting) latches (see Figure 3-49). These latches have outputs driven by tri-state buffers that supply 32 mA drive capability and can switch a 300 pF capacitive load in 22 ns (inverting) or 30 ns (non-inverting). The latches propagate the input signals through to the outputs while ALE is high and latch the outputs on the falling edge of ALE (see Figure 3-50). The outputs are enabled by the active-low $\overline{\text{OE}}$ input.

Chapter Four discusses other address strobe techniques that do not use the ALE signal from the 82288 Bus Controller. These special address strobes can be used to customize the 80286 bus cycle timing to accommodate particular memory or peripheral devices.

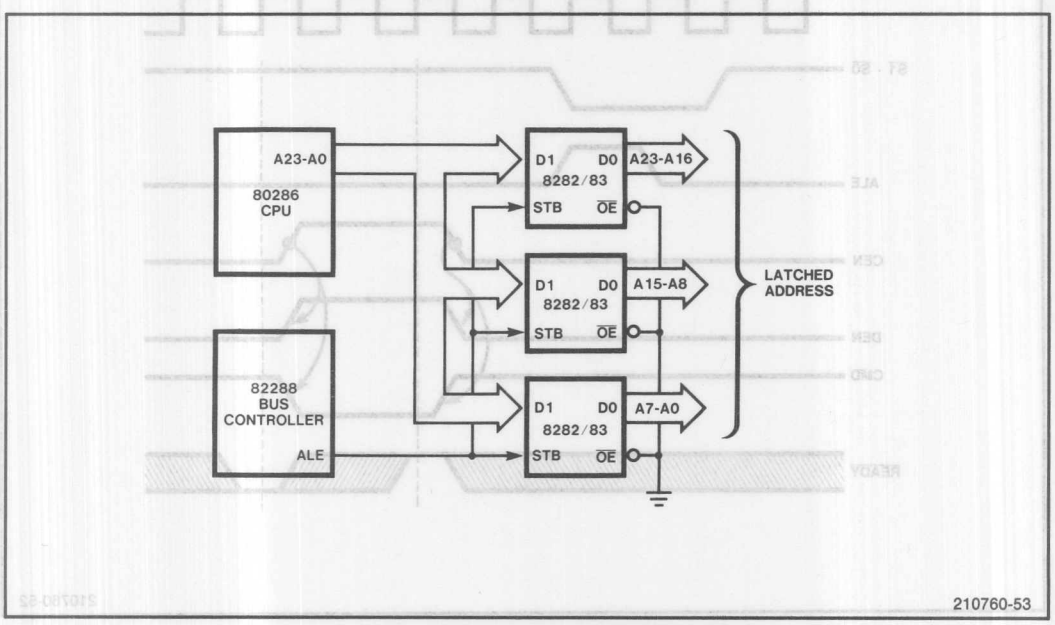


Figure 3-49. Latching the 80286 Address

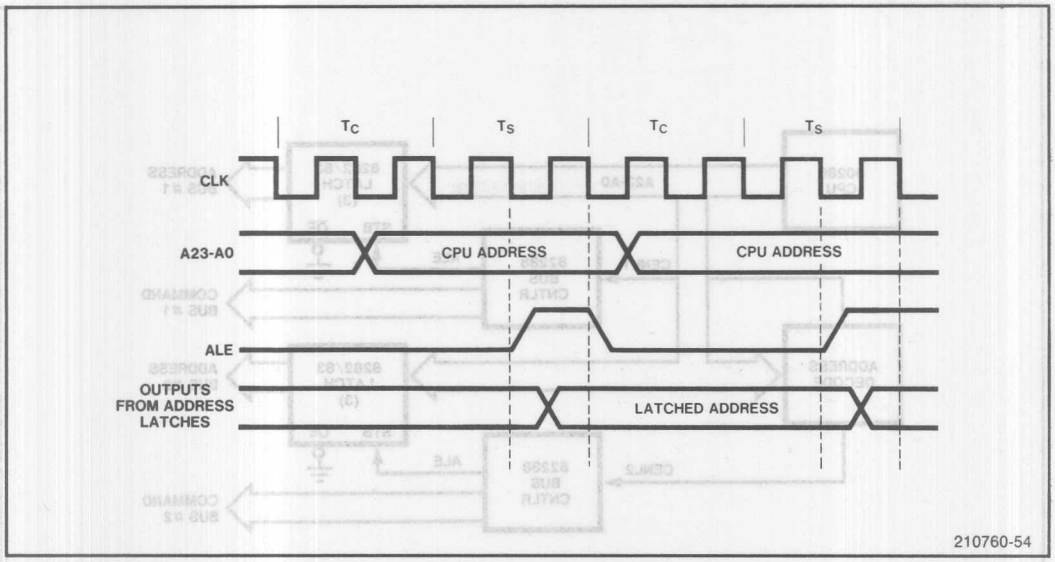


Figure 3-50. ALE Timing

For optimum system performance and compatibility with multi-processor Multibus systems, the address should be latched at the interface to each bus. As shown in Figure 3-51, systems that use both buffered local and public system buses can have separate address buses with separate latches and Controllers for each bus. The 100 pf maximum capacitance drive specification of the 80286 allows at least five sets of address latches to be connected to the address outputs of the 80286.

Address Decoding

Address decoding in an iAPX 286 system can be performed between the time that the address is generated at the CPU pins and the rising edge of ALE. At least 68 ns exist for an 8-Mhz system for this address decode logic to operate and produce a select output from the latches with the same timing as addresses. Such decode time comes free since it does not delay when a device can respond to an address. Select signals decoded from CPU address pins must be latched to guarantee that they will remain stable for the duration of the bus cycle.

Decoded select signals that drive the CENL input to the Bus Controller or the SYSB/$\overline{\text{RESB}}$ input to the Bus Arbiter do not need to be latched (both the Bus Controller and Bus Arbiter latch these signals internally).

NON-LATCHED CHIP SELECTS

Figure 3-52 shows an example circuit that decodes the address, M/$\overline{\text{IO}}$=0, and COD/$\overline{\text{INTA}}$=0 signals to provide non-latched CENL and SYSB/$\overline{\text{RESB}}$ selects for the Bus Controller and Bus Arbiter and additional non-latched selects to other devices in the system. The example assumes two system buses with all interrupts on bus #1. The M/$\overline{\text{IO}}$ and COD/$\overline{\text{INTA}}$ signals are used to enable bus #1 during interrupt-acknowledge cycles (the address bus floats during interrupt-acknowledge cycles).

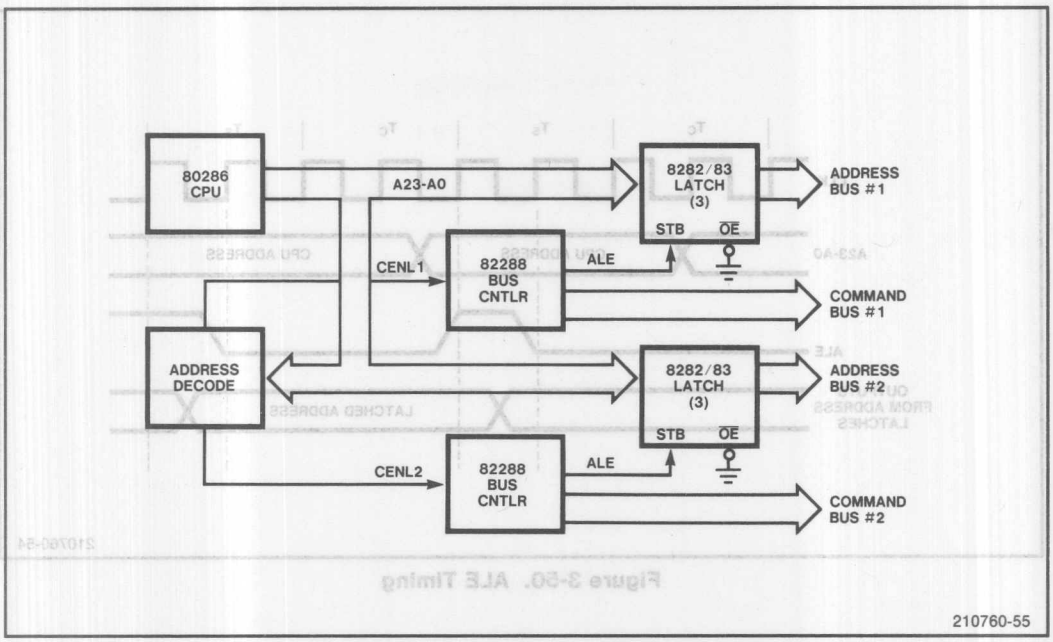
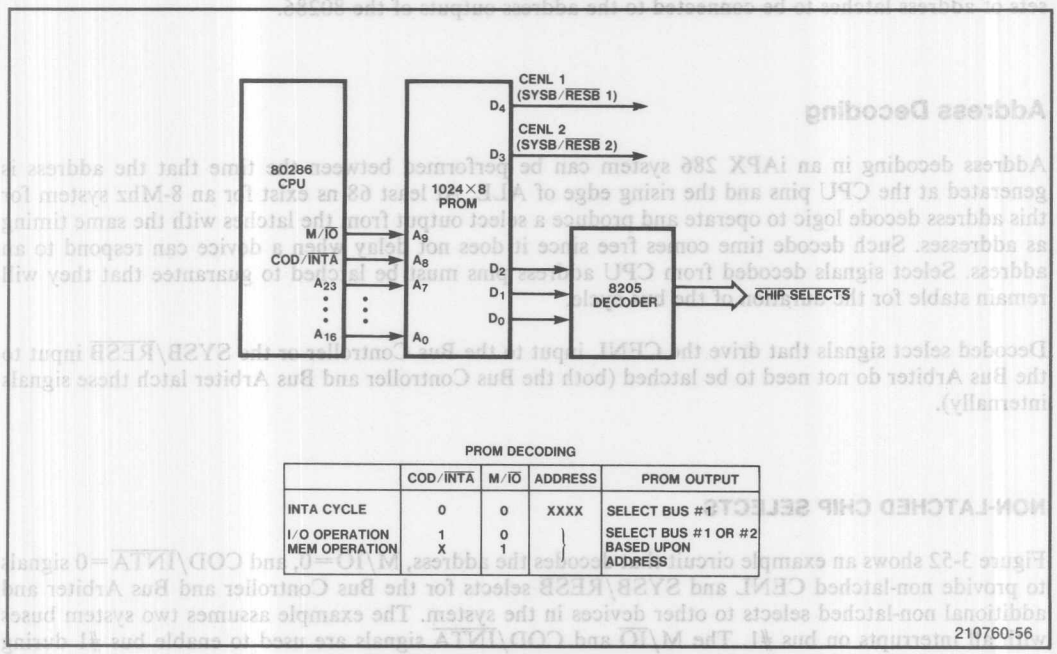


Figure 3-51. Dual Address Bus System



PROM DECODING

	COD/INTA	M/IO	ADDRESS	PROM OUTPUT
INTA CYCLE	0	0	XXXX	SELECT BUS #1
I/O OPERATION	1	0		SELECT BUS #1 OR #2 BASED UPON ADDRESS
MEM OPERATION	X	1		

Figure 3-52. Non-Latched Chip Selects

Set-up time for the Bus Controller CENL input is 20 ns before the start of T_c (assuming that commands are not delayed). This means that the time from A23-A0, M/$\overline{IO}$, and COD/$\overline{INTA}$ active to CENL is 107.5 ns max. To meet the set-up time for the Bus Arbiter SYS/$\overline{RESB}$ input, the time from A23-A0, M/$\overline{IO}$, and COD/$\overline{INTA}$ active to SYSB/$\overline{RESB}$ is 127.5 ns maximum. Since the same PROM output must meet both these requirements, the decoder PROM must have an access time of 107.5 ns or less.

To accommodate the timing requirements for individual devices, the 82288 Bus Controller has a CMDLY input to delay the leading edge of commands from the Bus Controller by one or more CLK cycles. Typical devices require zero or one command delays. This CMDLY (Command Delay) input to the Bus Controller is described in a previous section.

The CMDLY signal is typically generated by address-decode logic, and is not latched either externally or by the 82288. The same address-decoding techniques used for other non-latched selects can be used to generate zero or more command delays for individual devices or ranges of addresses. Figure 3-45 shown previously illustrates a method for generating zero- or one-CLK command delays based on a non-latched output from an address-decode PROM gated with ALE.

LATCHED CHIP SELECTS

Most memory and peripheral devices require a stable device-select signal that can be provided only by latched chip-selects. Latched chip-selects can easily be generated by adding latches to the outputs of decode logic shown in Figure 3-52. 8282/8283 or TTL devices can latch the selects in response to the ALE signal from the Bus Controller.

Figure 3-53 shows an example of a circuit that latches chip-selects. The timing for latched chip-selects differs from that of non-latched selects. Since ALE goes high only 68 ns after the 80286 address is valid, the address-decode PROM must have an access time of less than 68 ns to prevent the chip-select outputs from changing after being passed through the latches.

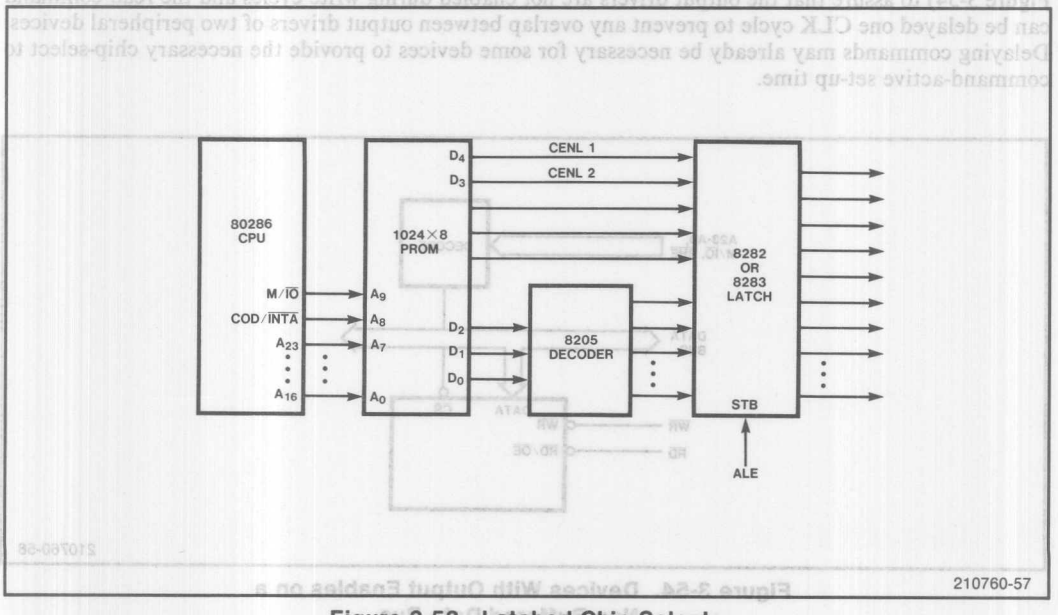


Figure 3-53. Latched Chip Selects

in to Strobe (ALE) going low, so a PROM with a maximum access time of up to 130 ns may actually be used. To provide sufficient set-up times from chip-select to command-active, however, commands from the Bus Controller may have to be delayed one CLK cycle when using this technique. By delaying commands one CLK cycle, the same circuit can be used with a gain of up to 62.5 ns in additional set-up time from chip-select to command active.

Depending on speed, cost, and available board space, the designer may substitute programmed array logic or discrete TTL logic for the PROM to decode latched and non-latched select signals from 80286 addresses.

Data Bus Interface

Since the 80286 provides separate address and data pins, demultiplexing of the data bus is unnecessary. The design considerations for connecting devices to the iAPX 286 local bus are whether or not to buffer the data bus and whether a single or double level of buffering is required for the intended application.

In general, buffering of the data bus is required for memory devices with $\overline{OE}\uparrow$ to data-bus-float times of 42 ns or more (to avoid contention with the 80286 during back-to-back read-write cycles). Most memory and I/O devices will therefore require buffering of the data bus. Unbuffered devices must not present a load exceeding the CPU's maximum drive capability of 3.0 mA with a capacitive loading of 100 pF.

A major concern in both buffered and non-buffered systems is the contention on the data bus that occurs when one device begins driving the bus before the previously-selected device has disabled its output drivers. Devices that have separate chip-select and output-enable or read inputs provide one solution to this problem. The output-enable signal can be driven by the Read command signal (see Figure 3-54) to assure that the output drivers are not enabled during write cycles and the read command can be delayed one CLK cycle to prevent any overlap between output drivers of two peripheral devices. Delaying commands may already be necessary for some devices to provide the necessary chip-select to command-active set-up time.

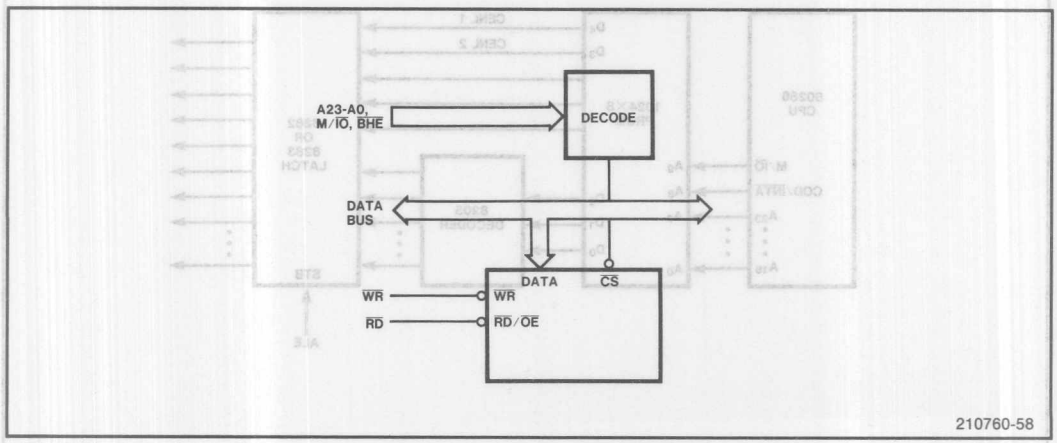


Figure 3-54. Devices With Output Enables on a Non-Buffered Data Bus

(Figure 3-55). The chip-select in this circuit is gated with Read and Write. The tradeoffs with this method include (1) chip select time is reduced to the read access time for reads and (2) no time is allowed for chip-select to write-command set-up and (3) no hold time from CS high or WE high for writes. Designers should check the device specifications to verify whether or not these tradeoffs create a problem for a specific case. In general, devices with separate output-enables are preferred.

To satisfy the capacitive loading and drive requirements of larger systems, the data bus must be buffered. The 8286 non-inverting and 8287 inverting octal transceivers satisfy the buffer requirements of most larger systems. 8286/8287 transceivers have three-state output buffers that drive 32 mA on the bus interface and 10 mA on the CPU interface and can switch capacitive loads of 300 pF on the bus interface (Side B) and 100 pF on the CPU interface (Side A) in 22 ns (8287) or 30 ns (8286). As shown in Figure 3-56, the $\overline{OE}$ and T inputs to the transceivers are driven by DEN and DT/$\overline{R}$ from the 82288 Bus Controller. DEN enables the devices, while DT/$\overline{R}$ controls the direction of data through the devices. The $\overline{DISABLE}$ term is used if any other device may drive local data bus (i.e., 80287) instead of data transceivers. Chapter Six describes using the 80287 in an iAPX 286/20 system, and describes how to control the local data transceivers.

For applications that require separate system buses, two or more sets of transceivers can be connected to the 80286 data pins. Figure 3-57 shows a configuration having two separate buses. Each bus has its own 82288 Bus Controller to control its own set of data transceivers. Since the address-decode logic selects one bus at a time, only one of the two DEN signals will ever be active at any time during a read cycle.

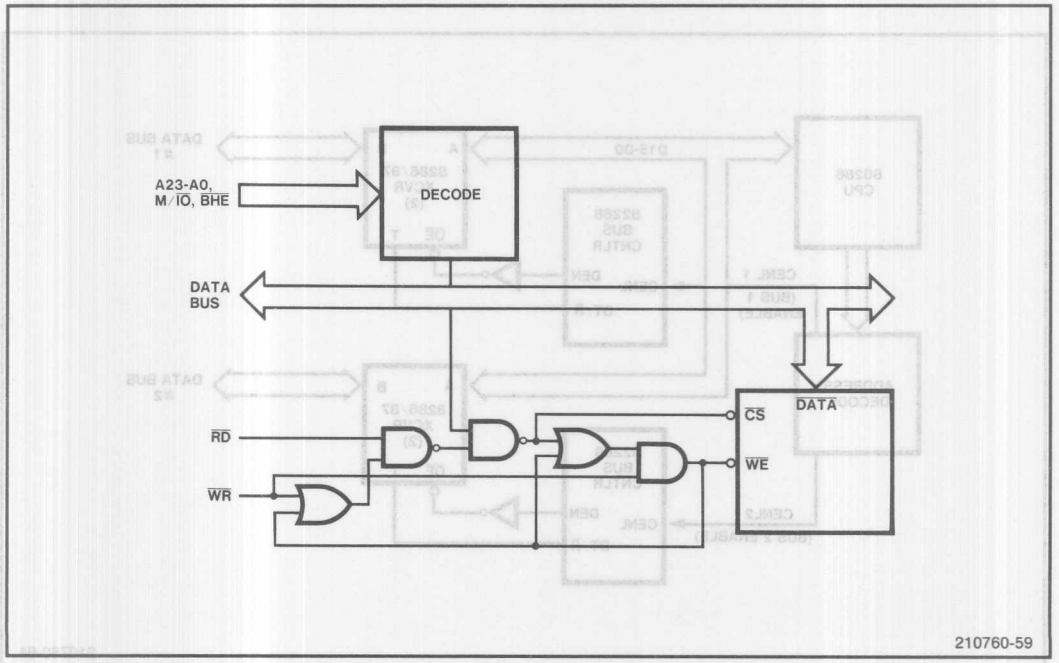


Figure 3-55. Devices Without Output Enables on the Local Data Bus

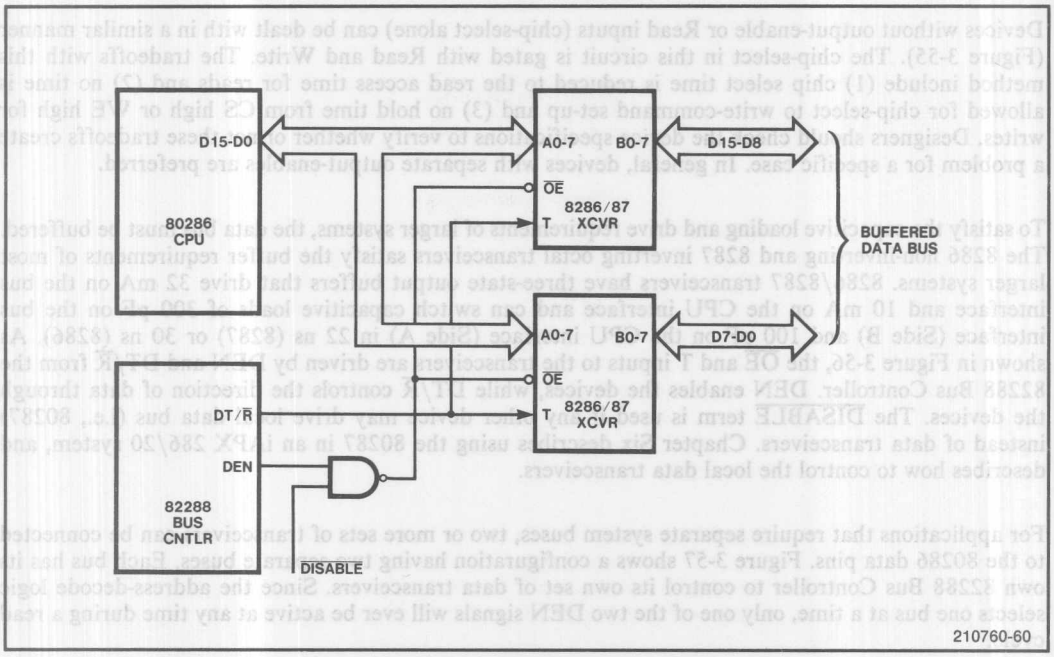


Figure 3-56. Buffering the Data Bus

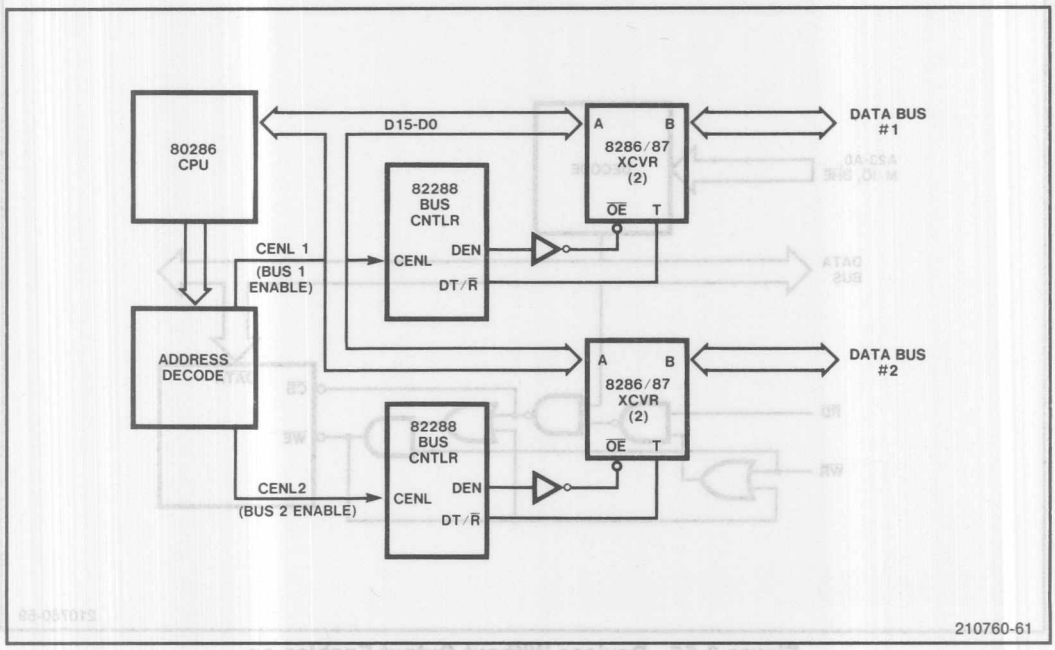


Figure 3-57. Dual Data Bus System

Another alternative when implementing a data bus is to add a second level of buffering, reducing the load seen by peripheral devices connected to the system bus (Figure 3-58). Double-buffering is typically used for multi-board systems and isolation of memory arrays. The concerns with this configuration are the additional delay for access, and more importantly, control of the second level of transceivers in relationship to the system bus and the device being interfaced to the system bus. Several techniques can be used to control these transceivers.

The first technique (shown in Figure 3-59) simply distributed DEN and DT/$\overline{R}$ throughout the system. DT/$\overline{R}$ is inverted to provide proper direction control for the second level transceivers, since sides A and B were reversed. The second example (shown in Figure 3-60) provides control for devices with output enables. The buffers are selected whenever the device on the local bus is selected. $\overline{RD}$ directs the data to the local bus device during read cycles.

Bus contention on the local bus is possible during a read as $\overline{RD}$ simultaneously enables the device output and changes the transceiver direction. Contention may also occur when $\overline{RD}$ terminates. If $\overline{OE}$ was active before T changes, then the output is the same as the input, with no contention. To eliminate such contention, it is necessary to sequence T and $\overline{OE}$.

For devices without output enables, the same technique can be applied if the chip select to the device is conditioned by read or write signals (see Figure 3-61). Controlling the chip select with read/write prevents the device from driving against the transceiver prior to the command being received. This technique limits access time to read/write time and chip select to write set-up and hold times.

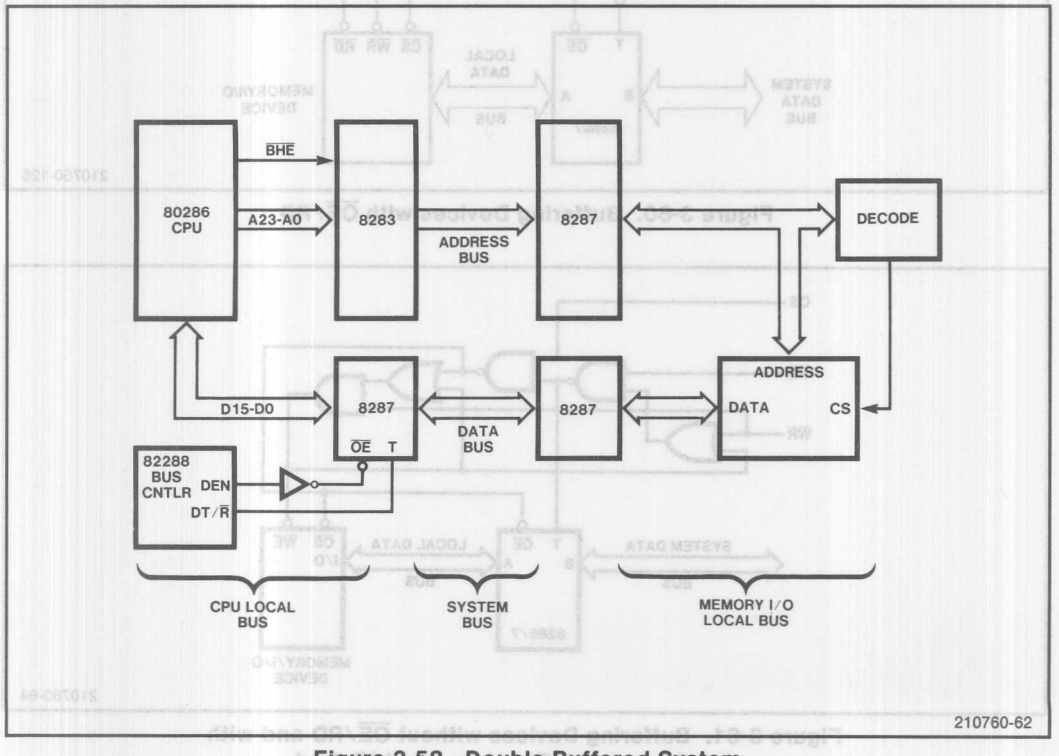


Figure 3-58. Double Buffered System

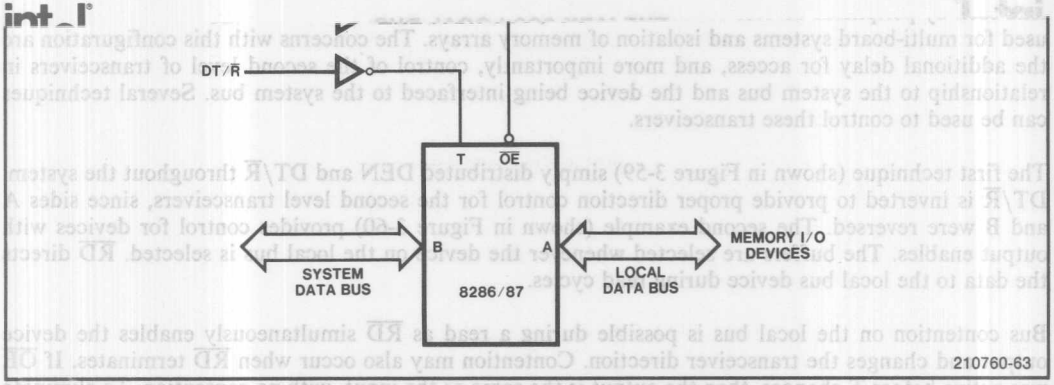


Figure 3-59. Controlling System Transceivers with DEN and DT/$\overline{R}$

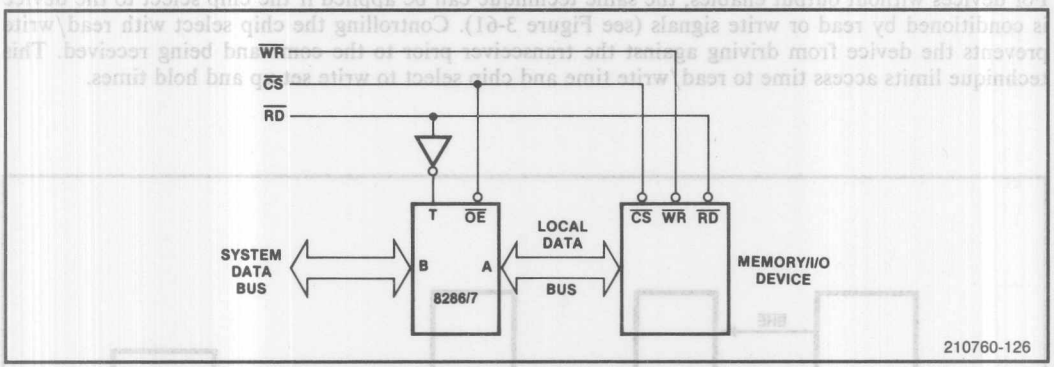


Figure 3-60. Buffering Devices with $\overline{OE}$/RD

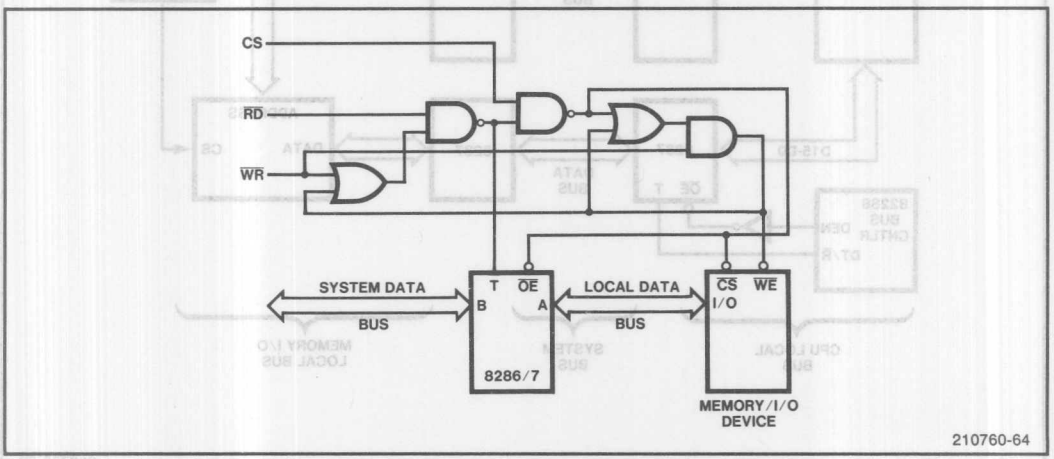


Figure 3-61. Buffering Devices without $\overline{OE}$/RD and with Common or Separate Input/Output

One last technique is given for devices with separate inputs and outputs (see Figure 3-62). Separate receivers and drivers are provided, rather than a single transceiver. The receiver is always enabled while the bus driver is controlled by $\overline{RD}$ and chip-select. The only possibility for bus contention in this system occurs as multiple devices on each line of the local read bus are enabled and disabled during chip selection changes.

Other Bus Masters on the iAPX 286 Local Bus

The 80286 provides on-chip arbitration logic that supports a protocol for transferring control of the local bus to other bus masters. This protocol is implemented by a pair of handshake signals called hold (HOLD) and hold acknowledge (HLDA). The sequence of signals shown in Figures 3-63 and 3-64 illustrate this protocol. To gain control of the local bus, the requesting bus master asserts an active high signal on the 80286 HOLD input (Figure 3-63). This HOLD input need not be synchronous to the 80286 CLK.

When HOLD is asserted, the 80286 completes the current bus sequence and then tri-states all bus outputs (except HLDA) to effectively remove itself from the local bus. The 80286 then drives HLDA high to signal the requesting bus master that it now has control of the bus. The requesting bus master must maintain a high on the HOLD line until it no longer requires the local bus.

The hold request to the CPU affects the 80286 Bus Unit (BU) directly, and only indirectly affects the other units. The Execution Unit continues to execute from the instruction queue until both the instruction queue and the pre-fetch queue are empty, or until an instruction is encountered that requires a

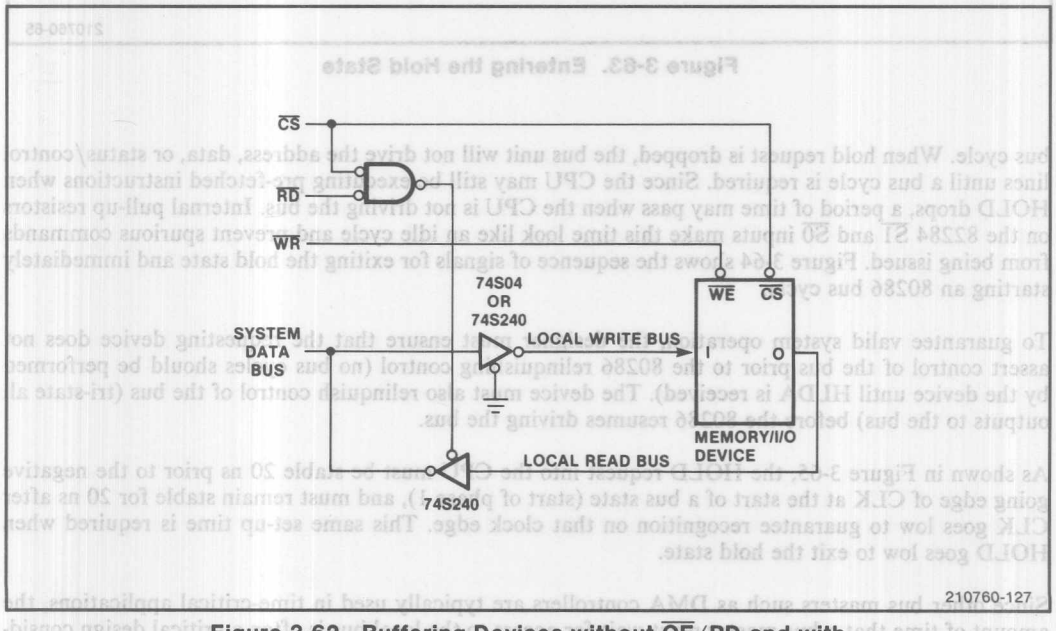


Figure 3-62. Buffering Devices without $\overline{OE}$/RD and with Separate Input/Output

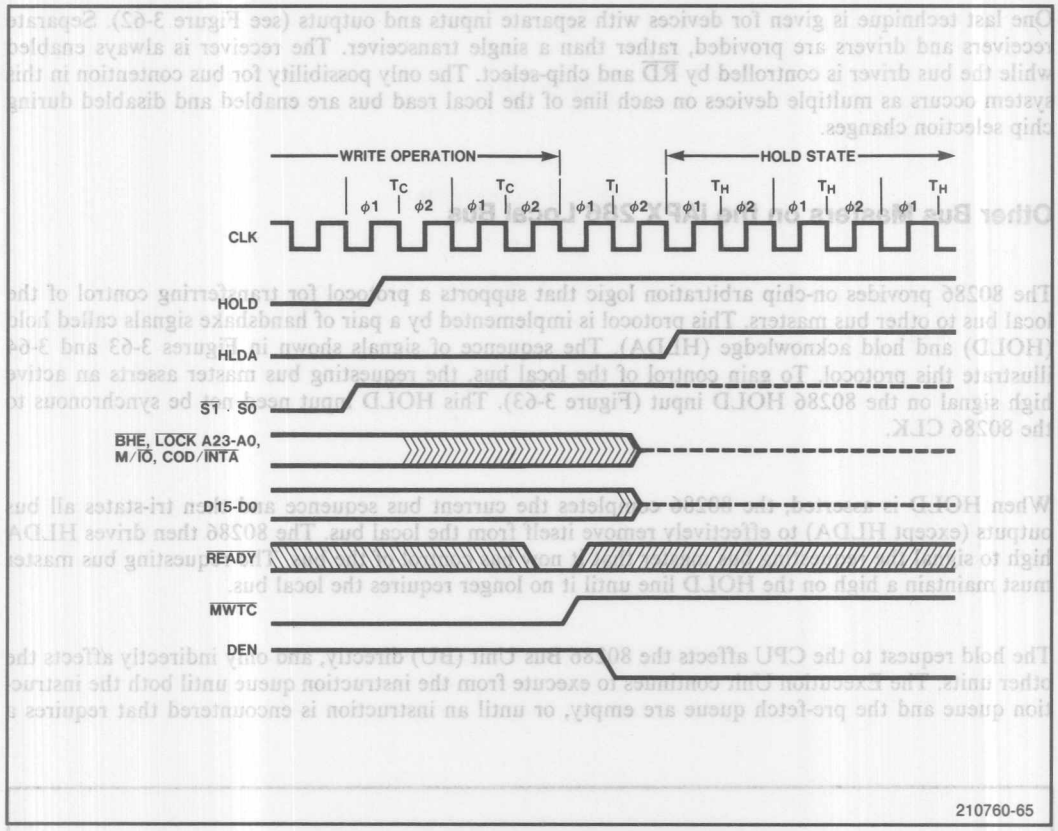


Figure 3-63. Entering the Hold State

bus cycle. When hold request is dropped, the bus unit will not drive the address, data, or status/control lines until a bus cycle is required. Since the CPU may still be executing pre-fetched instructions when HOLD drops, a period of time may pass when the CPU is not driving the bus. Internal pull-up resistors on the 82284 $\overline{S1}$ and $\overline{S0}$ inputs make this time look like an idle cycle and prevent spurious commands from being issued. Figure 3-64 shows the sequence of signals for exiting the hold state and immediately starting an 80286 bus cycle.

To guarantee valid system operation, the designer must ensure that the requesting device does not assert control of the bus prior to the 80286 relinquishing control (no bus cycles should be performed by the device until HLDA is received). The device must also relinquish control of the bus (tri-state all outputs to the bus) before the 80286 resumes driving the bus.

As shown in Figure 3-65, the HOLD request into the CPU must be stable 20 ns prior to the negative going edge of CLK at the start of a bus state (start of phase 1), and must remain stable for 20 ns after CLK goes low to guarantee recognition on that clock edge. This same set-up time is required when HOLD goes low to exit the hold state.

Since other bus masters such as DMA controllers are typically used in time-critical applications, the amount of time that a bus master must wait for access to the local bus is often a critical design consideration.

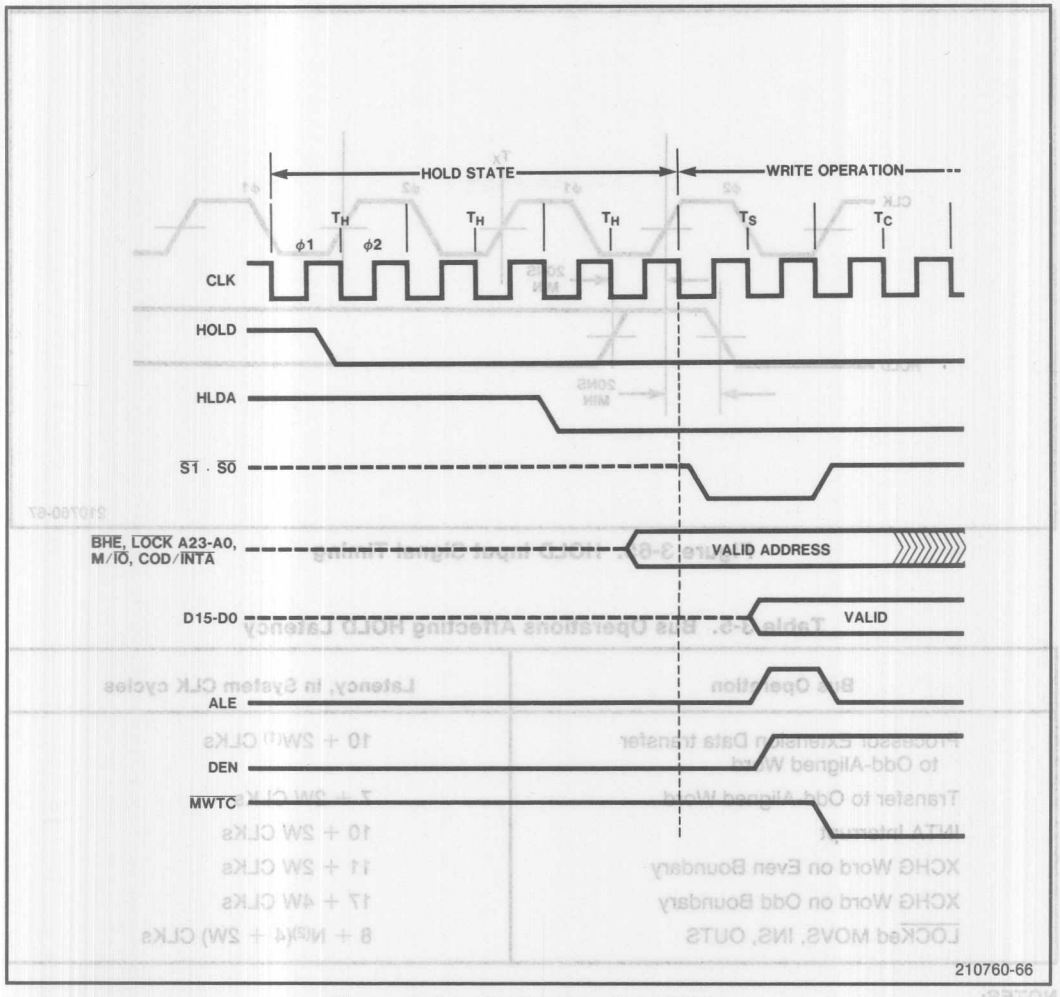


Figure 3-64. Exiting the Hold State

Figures 3-63 and 3-64 show the minimum possible latency for the 80286 CPU (2½ processor clocks from HOLD active to HLDA active and 1½ processor clocks from HOLD inactive to HLDA in active).

The maximum possible HOLD-active to HLDA-active time (HOLD latency) depends on the software being executed. The actual HOLD latency at any time depends on the current bus activity, the state of the $\overline{\text{LOCK}}$ signal (internal to the CPU) activated by the $\overline{\text{LOCK}}$ prefix, and interrupts. The 80286 will not honor a HOLD request until the current bus operation is complete. Table 3-5 shows the types of bus operations that can affect HOLD latency, and indicates the types of delays that these operations may introduce. When considering maximum HOLD latencies, designers must select which of these bus operations are possible, and then select the maximum latency from among them.

As indicated in Table 3-5, wait states affect HOLD latency. The 80286 CPU will not honor a HOLD request until the end of the current bus operation, no matter how many wait states are required. Systems with DMA where data transfer is critical must insure that $\overline{\text{READY}}$ returns sufficiently soon.

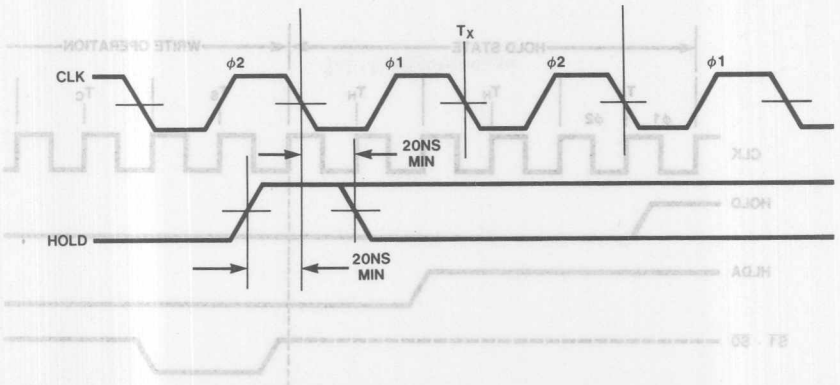


Figure 3-65. HOLD Input Signal Timing

Table 3-5. Bus Operations Affecting HOLD Latency

Bus Operation	Latency, in System CLK cycles
Processor Extension Data transfer to Odd-Aligned Word	10 + 2W(1) CLKs
Transfer to Odd-Aligned Word	7 + 2W CLKs
INTA Interrupt	10 + 2W CLKs
XCHG Word on Even Boundary	11 + 2W CLKs
XCHG Word on Odd Boundary	17 + 4W CLKs
$\overline{\text{LOCK}}$ed MOVS, INS, OUTS	8 + N(2)(4 + 2W) CLKs

NOTES:

(1) W = number of Wait states per cycle

(2) N = number of transfers

DMA CONFIGURATION

A typical use of the HOLD/HLDA signals in an iAPX 286 system is bus control exchange with DMA controllers or I/O devices that perform DMA. Figure 3-66 shows a block-level inter-connect diagram for a "generic" DMA controller. The DMA controller resides on the local bus. This is probably the simplest DMA configuration possible with the 80286, but it requires that the DMA controller have an interface that functions like the 80286 local bus signals.

DMA controllers that do not have an 80286-like interface may still be configured in an iAPX 286 system at the system bus level. Figure 3-67 shows a general inter-connect diagram for this type of application. The CPU is totally isolated from the system bus when in the T_h state, and the DMA controller must be compatible with the iAPX 286 system bus.

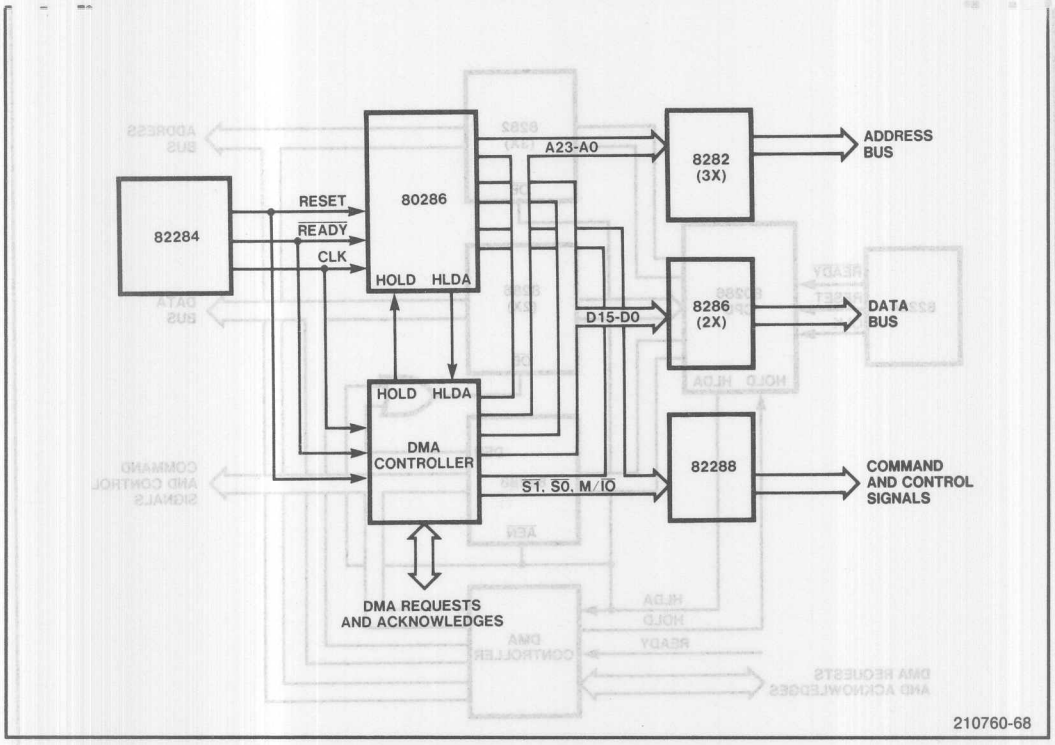


Figure 3-66. DMA Controller on the Local Bus

Initializing the iAPX 286 Processor

The 80286 RESET input provides an orderly way to start or restart a system. When the processor detects the positive-going edge of a pulse on RESET, it terminates all activities until the signal goes low, at which time it initializes the CPU to a known internal state; the CPU then begins fetching instructions from absolute address FFFFF0H.

80286 INTERNAL STATES

When an active RESET signal goes low, the 80286 registers are initialized as shown in Table 3-6. The valid fields of the segment and description registers are set on, indicating valid segments. All privilege-level fields are set to zero.

The RESET signal initializes the CPU in Real-Address mode, with the CS base register containing FF0000H and IP containing FFF0H. The first instruction fetch cycle following reset will be from the physical address formed by these two registers, i.e., from address FFFFF0H. This location will normally contain a JMP instruction to the actual beginning of the system bootstrap program.

For iAPX 286 systems to operate in Protected Virtual-Address mode, the 80826 (executing in Real-Address mode) must enter Protected mode as part of the software initialization routine.

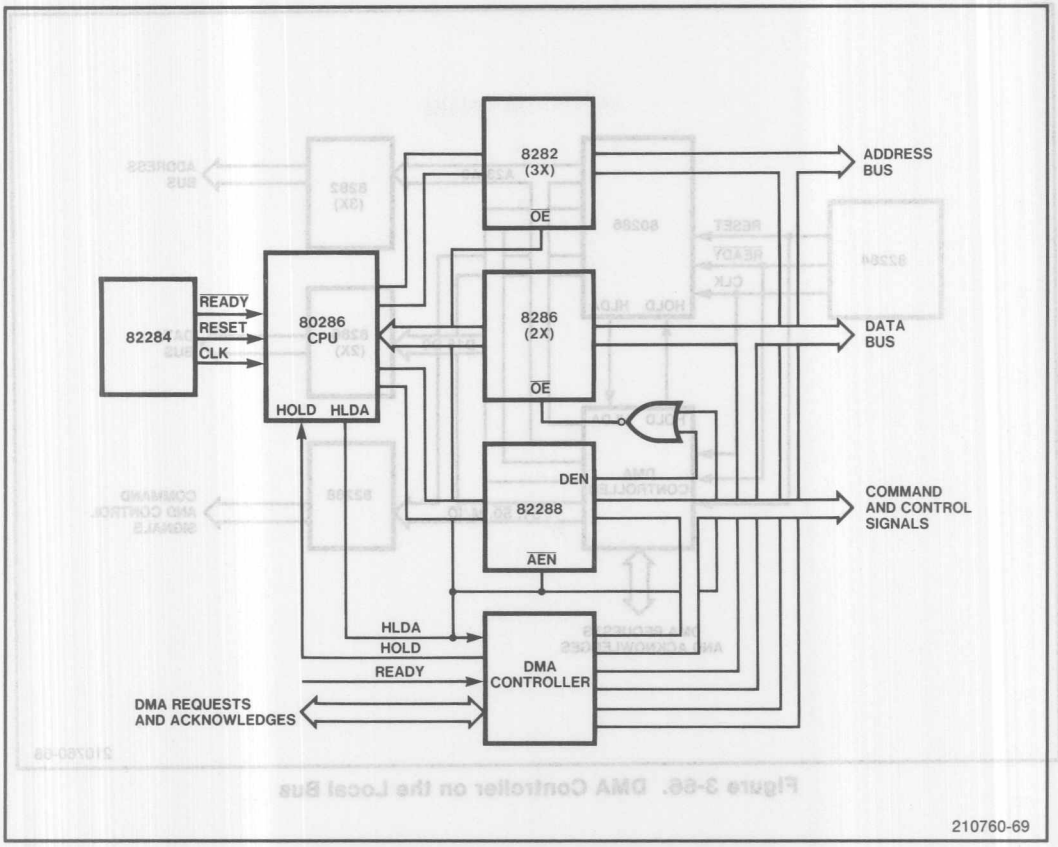


Figure 3-67. DMA Controller on a Private System Bus

Table 3-6. CPU State Following RESET

CPU Component	Content
Flags	0002H
MSW	FFF0H
IP	FFF0H
CS Selector	F000H
DS Selector	0000H
SS Selector	0000H
ES Selector	0000H
CS Base	FF0000H
DS Base	000000H
SS Base	000000H
ES Base	000000H
CS Limit	FFFFH
DS Limit	FFFFH
SS Limit	FFFFH
ES Limit	FFFFH
IDT Base	000000H
IDT Limit	FFFFH

To accommodate an 80286 operating in both Real-Address mode and Protected mode, the EPROMs containing the system bootstrap program must answer to both a 20-bit and a 24-bit physical address. In Real-Address mode, the system bootstrap EPROMs must respond to addresses in the available 1-Megabyte address space (ignoring the upper four address bits). In Protected mode, these same EPROMs typically respond to addresses only in the top megabyte of the available 16-Megabyte address space (using the full 24-bit address).

Figure 3-68 shows a circuit that permits this type of operation by generating one of the terms in the address-decode logic selecting the bootstrap EPROMs. This term inhibits the decoding of A23-A20 after RESET, when the system runs in Real-Address mode. After entering Protected mode, the bootstrap program must strobe the $\overline{\text{PROTMODE}}$ signal to allow full use of the available 16-Megabyte address space.

80286 EXTERNAL SIGNALS

At power-up, the 80286 CPU requires 5 milliseconds to allow the capacitor connected to the CAP pin to charge up. RESET should be asserted during this time to prevent spurious outputs. After power-up, the CPU requires a high on the RESET input with a minimum pulse width of 16 processor clocks. The processor is internally active for a minimum of 50 CLK cycles after RESET goes low before performing the first memory cycle. Maskable and non-maskable interrupts (the INT and NMI inputs) are not recognized during the internal reset.

When RESET is driven high, the 80286 signals will enter the states shown in Table 3-7. The timing for the signals during reset is shown in Figure 3-69.

The 80286 data bus lines enter the tri-state OFF condition when RESET is active. If system RESET occurs during a bus cycle, RESET forces the 82284 Clock Generator to drive $\overline{\text{READY}}$ low to terminate the bus cycle and reinitialize the 82288 Bus Controller.

The 82284 Clock Generator contains internal pull-up resistors on its $\overline{\text{S1}}$ and $\overline{\text{S0}}$ status inputs, causing the Bus Controller to interpret the 80286 RESET condition as an idle bus state. The outputs from the Bus Controller during an idle state are shown in Table 3-8.

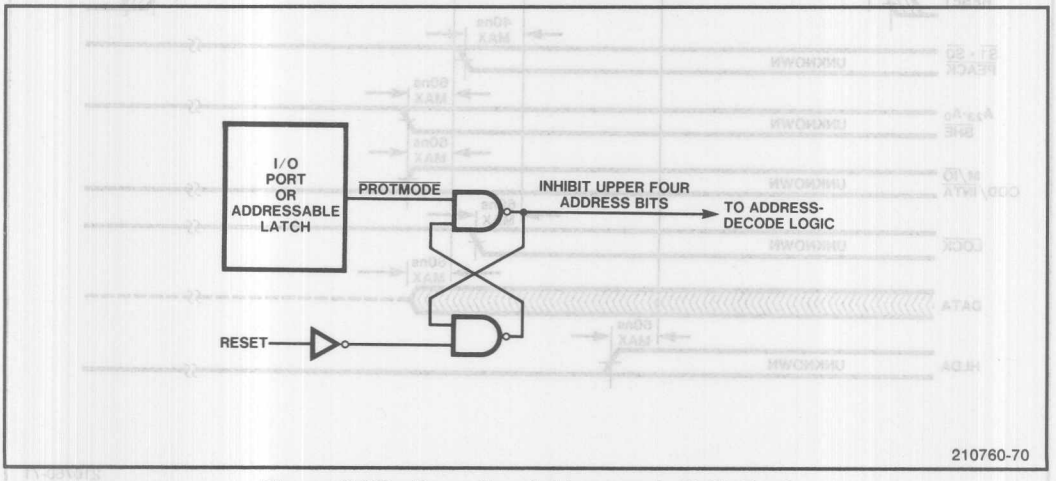


Figure 3-68. Decoding Addresses in Both Real and Protected Modes

Signals	Condition
A23-A0	Logic 1
D15-D0	Tri-state OFF
$\overline{S1}$,$\overline{S0}$	Logic 1
$\overline{PEACK}$	Logic 1
$\overline{BHE}$	Logic 1
$\overline{LOCK}$	Logic 1
M/$\overline{IO}$	Logic 0
COD/$\overline{INTA}$	Logic 0
HLDA	Logic 0

Table 3-8. 82288 Command States During RESET

Signal	State
ALE	Logic 0
DEN	Logic 0
DT/$\overline{R}$	Logic 1
MCE	Logic 0
Commands	Logic 1

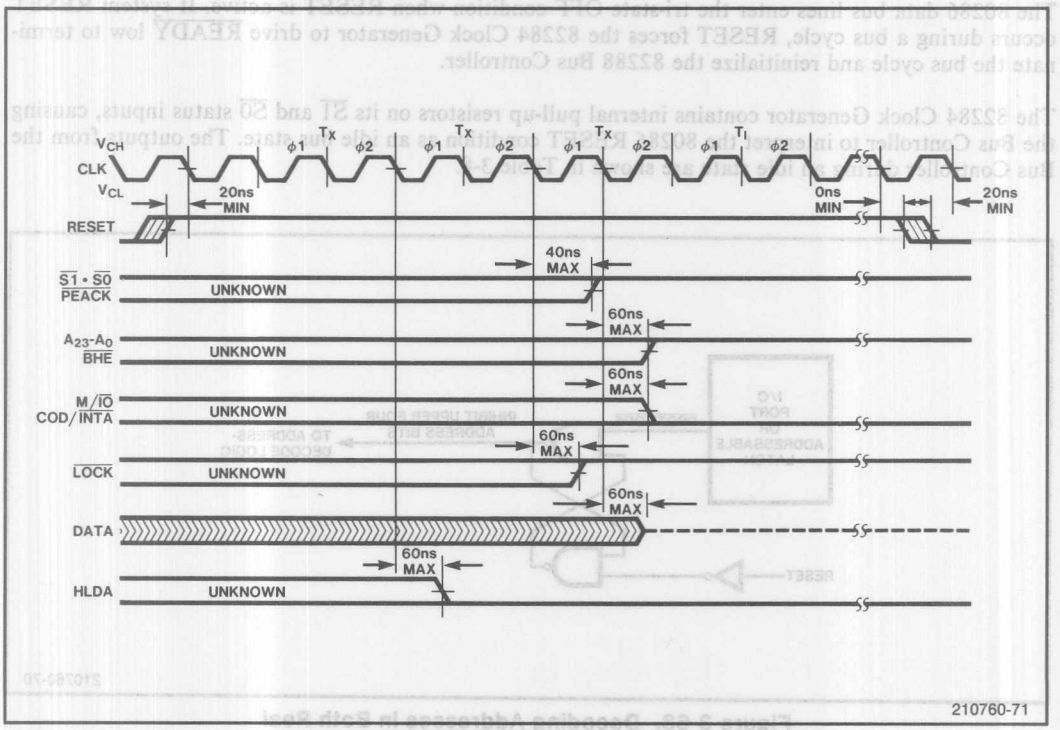


Figure 3-69. Signal States During RESET

Note that the 82288 Bus Controller does not tri-state its command outputs during idle bus states. In a single CPU system, if the system designer wishes to effectively remove the 80286 CPU from the bus during RESET, the RESET signal can be connected to the 82288's MB input and to the output enable of the address latches (Figure 3-70). This forces the command and address bus interface into the tri-state OFF condition while the 82288 drives DEN inactive, disabling the data bus transceivers. If the 82288 command outputs are tri-stated during RESET, the command lines should be pulled to V_{cc} through 2.2K ohm resistors to keep these signals in the HIGH state.

iAPX 286 Bus Timing

Figure 3-71 shows the timing relationships of the major iAPX 286 local bus cycles. Included in this figure are the timing relationships for:

- Address and ALE timing
- Command timing
- $\overline{\text{READY}}$ timing
- Read cycle timing
- Write cycle timing

For most of these signals, timing is controlled by the system CLK signal. For this reason, the timing relationships between signals can be deduced simply by determining the clock cycles between the controlling clock edges, and adding or subtracting the appropriate minimum or maximum timing delays. Additional delays through any transceivers or latches must also be considered when determining signal timing.

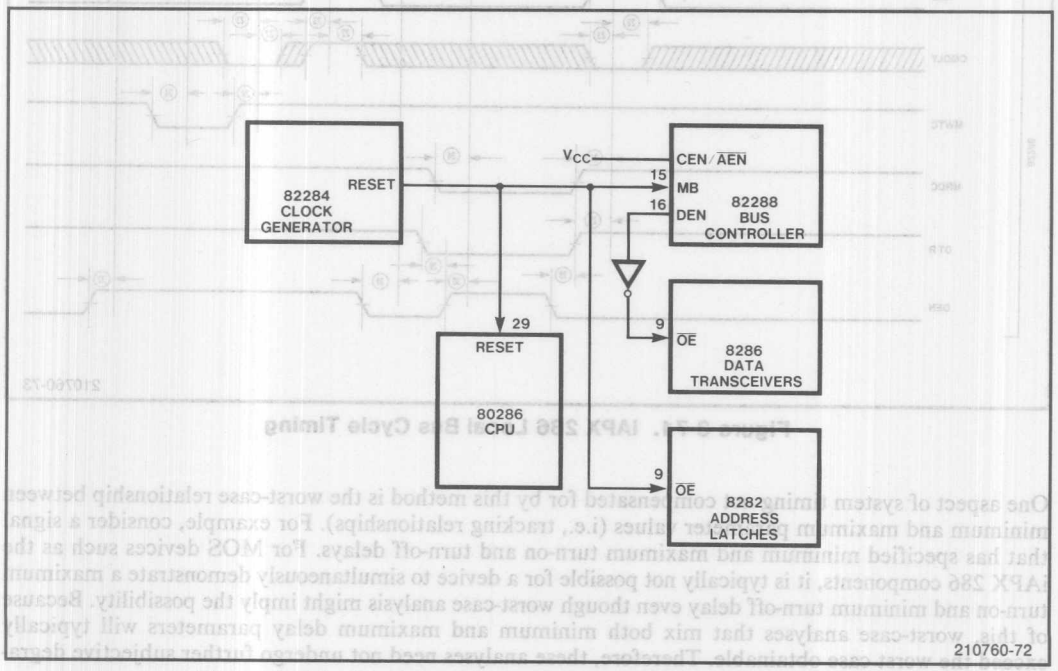


Figure 3-70. Disabling the 80286 Bus Interface on RESET

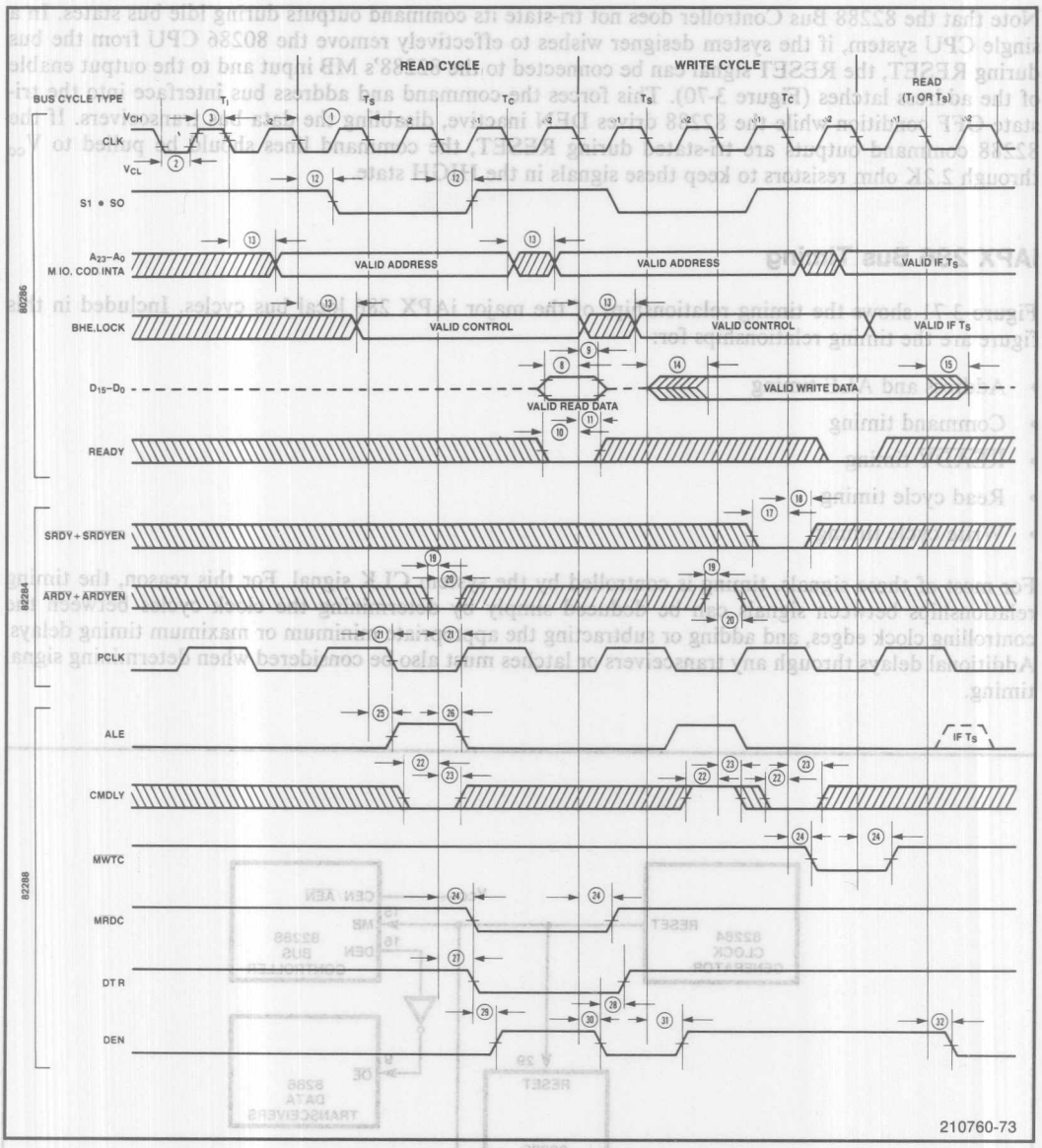


Figure 3-71. iAPX 286 Local Bus Cycle Timing

One aspect of system timing not compensated for by this method is the worst-case relationship between minimum and maximum parameter values (i.e., tracking relationships). For example, consider a signal that has specified minimum and maximum turn-on and turn-off delays. For MOS devices such as the iAPX 286 components, it is typically not possible for a device to simultaneously demonstrate a maximum turn-on and minimum turn-off delay even though worst-case analysis might imply the possibility. Because of this, worst-case analyses that mix both minimum and maximum delay parameters will typically exceed the worst case obtainable. Therefore, these analyses need not undergo further subjective degradation to obtain reliable worst-case values.

The following sections provide guidelines for analyzing the critical 80286 timing relationships for each of the signal groups described above. In these sections, an 8-MHz 80286 is assumed (one system CLK cycle = 62.5 ns). 80286 processors running at other clock frequencies will have correspondingly shorter or longer values for each of their timing relationships.

For a more detailed discussion of local bus timing related specifically to interfacing memory and peripheral devices to the iAPX 286, see Chapters Four and Five.

ADDRESS AND ALE TIMING

The address/ALE timing relationship is important to determine the ability to capture an address and local chip-selects during a bus cycle. The following discussion considers the standard bus timing using the ALE signal from the 82288 Bus Controller. Other design techniques having different timing relationships are described in Chapter Four.

The maximum address valid delay ($(13)_{max}$ = 60 ns) from the 80286 guarantees an address set-up time of at least 68 ns before ALE goes active, even assuming a minimum ALE active delay ($(25)_{min}$ = 3 ns). If we assume a maximum strobe to output delay for the latches (STB-to-Output delay (max) = 45 ns for 8282 and 40 ns for 8283), and a maximum ALE active delay ($(25)_{max}$ = 12 ns), then the latest a valid address will be available at the outputs to the latches would be 5.5 ns before the start of T_c for the 8282 latch, and 10.5 ns for the 8283 latch.

The minimum address hold time from ALE inactive (42.5 ns) guarantees a stable address until well after ALE goes low. The ALE and latched address hold times guarantee a valid address to the system for a minimum of 50 ns after command inactive.

Note that the maximum valid delay and minimum hold time for $\overline{BHE}$ guarantee that it will be latched by ALE and will be available to the system at the same time and for the same duration as the latched address.

ALE may also be used to capture latched chip-selects for local memory or I/O. To provide valid selects before ALE falls low, address decoding must be performed in the window between address valid and the trailing edge of ALE:

3 CLK cycles at 8 MHz	187.2 ns
− Address valid delay (max)	− 60.0 ns
− ALE inactive delay (min)	− 0.0 ns
− Input-to-STB setup time (min)	− 0.0 ns
Minimum address decode time	127.5 ns

Decoding should be performed sooner if a device select is required before a command signal goes active. Latched chip-selects are described in a previous section.

COMMAND TIMING

Command timing in Figure 3-71 is shown for an $\overline{MRDC}$ command with no delays, and for an $\overline{MWTC}$ command with 1 delay. Commands are typically delayed using the 82288 CMDLY input to provide a longer time from address and chip-select valid to command active. The use of CMDLY was discussed in a previous section. Note that delaying a command does not automatically lengthen the bus cycle.

identical (Command delay (min) = 3 ns), and maximum turn-on and turn-off times are also the same (Command delay (max) = 20 ns). Thus, the minimum command pulse width is:

2 CLK cycles at 8 MHz	125 ns
− Command turn-on delay (max)	− 20 ns
+ Command turn-off delay (min)	+ 3 ns
Minimum Command Pulse Width =	108 ns

with no delay and 45.5 ns with 1 delay. These command pulse widths are increased by 125 ns (2 CLK cycles) for each wait state inserted into a bus cycle.

READY TIMING

The detailed requirements of the 80286 $\overline{\text{READY}}$ signal and the Ready inputs for the 82284 were described in an earlier section. The 82284 Ready inputs are typically generated from the decoded address for a selected device, or address decode and command signals. Ready timing is shown for 0 wait state execution to illustrate the relationship of the 82284 $\overline{\text{ARDY}}$ and $\overline{\text{SRDY}}$ input requirements with respect to address decode outputs and command signals.

Outputs from address decode logic can easily meet the $\overline{\text{ARDY}}$ or $\overline{\text{SRDY}}$ input requirements. $\overline{\text{ARDY}}$ obviously cannot be qualified by a command signal, however, even when the command is not delayed. A command with no delay will meet the set-up requirements of $\overline{\text{SRDY}}$. Once the maximum command active delay of 25 ns and the minimum $\overline{\text{SRDY}}$ set-up time of 20 ns are subtracted from phase 1 of the T_c clock cycle (62.5 ns), 17.5 ns remains for external logic to generate the $\overline{\text{SRDY}}$ signal. Commands with 0 or 1 delays may be used to qualify either ready input if the system is running with one or more wait states. Assuming the maximum command active delays, the external logic times for zero or one wait state operation at 8 MHz are as follows:

82284 Input	Set-up Time	Hold Time	External Logic Time 0 Cmd Delay	External Logic Time 1 Cmd Delay
$\overline{\text{ARDY}}$	0 ns	16 ns	100 ns	37.5 ns
$\overline{\text{SRDY}}$	20 ns	0 ns	142.5 ns	80 ns

An additional 125 ns (2 CLK cycles) are added to external logic time for each additional wait state.

READ CYCLE TIMING

Read timing consists of conditioning the bus, activating the read command, and establishing the data transceiver enable and direction controls. Figure 3-71 shows read timing for a CPU running with 0 wait states. During a read, the latched address is available to the system a minimum of 2.5 ns before the start of T_c, or 7.5 ns before command active. The read command is driven active early in T_c to enable the addressed device. DT/$\overline{\text{R}}$ is also driven low to condition the data transceivers to receive data, and DEN then enables the transceivers.

Data from the selected device must be valid at the CPU 10 ns before the end of T_c and must hold valid for 5 ns after T_c. Any propagation delay through data transceivers must also be considered. Maximum input to output delay through the 8286 transceiver is 30 ns; maximum delay through the 8287 is 22 ns. This results in the following minimum address valid to data valid time (using 8286 non-inverting transceivers):

3 CLK cycles at 8 MHz	187.5 ns
− ALE active delay (max)	− 15.0 ns
− 8282 STB-to-output delay (max)	− 45.0 ns
− 8286 transceiver delay (max)	− 30.0 ns
− Read data setup (min)	− 10.0 ns
Minimum Address access time =	87.5 ns

Minimum command active to read data valid time is 90 ns. These timing relationships increase by 125 ns for each wait state added to the bus cycle, as follows:

Minimum Times	0 Wait States	1 Wait State	2 Wait States	3 Wait States
Address valid to Buffered Data Valid (using 8286)	87.5 ns	212.5 ns	337.5 ns	462.5 ns
Address valid to Data valid (unbuffered)	117.5 ns	242.5 ns	367.5 ns	492.5 ns
CMD active to Data Valid	90 ns	215 ns	340 ns	465 ns

Note that the Address-valid to buffered-data-valid times assume an 8286 device; for the inverting 8287, these minimum times can be increased by 8 ns each.

WRITE CYCLE TIMING

Write timing involves providing write data to the system, generating the write command, and controlling data bus transceivers. Figure 3-71 shows write timing for a CPU running with 0 wait states. The transceiver direction control signal $DT/\overline{R}$ is conditioned to transmit at the end of each read cycle and does not change during a write cycle ($DT/\overline{R}$ is shown going high after a read). Data is placed on the data bus and DEN is driven active early in the bus cycle. Write data is guaranteed to be valid before the start of T_c (12.5 ns min), and before the write command becomes active (17.5 ns min). The write command is enabled early in T_c. Following is minimum time from write data valid until the data is latched into the system device (command inactive):

3 CLK cycles at 8 MHz	187.5 ns
− Write-data valid delay (max)	− 50.0 ns
+ Command inactive delay (min)	+ 3.0 ns
Minimum write data access time =	140.5 ns

Write command pulse width is 108 ns minimum. Data is held valid for a minimum of 37.5 ns after the write command goes inactive. All of these timing relationships increase 125 ns for each added wait state, as follows:

Time	0 Wait States	1 Wait State	2 Wait States	3 Wait States
Data Valid to CMD inactive	142.5 ns	267.5 ns	392.5 ns	517.5 ns
Address valid to CMD inactive	132.5 ns	257.5 ns	382.5 ns	507.5 ns
Command Pulse Width	108 ns	233 ns	358 ns	473 ns

Of course, any propagation delays through data transceivers should be considered when finding the actual data set-up times for the system device being written to. The maximum input to output delay through the 8286 data transceiver is 30 ns; maximum delay through the inverting 8287 is 22 ns.

INTERRUPT-ACKNOWLEDGE TIMING

Timing for an interrupt-acknowledge cycle is identical to a read, with the $\overline{\text{INTA}}$ command being issued instead of $\overline{\text{MRDC}}$ or $\overline{\text{IORC}}$. The MCE signal is also issued during an interrupt-acknowledge cycle, going high following the start of Phase 2 of Ts during the first INTA cycle, and falling low during Phase 2 of the final T_c of the cycle. The two back-to-back $\overline{\text{INTA}}$ cycles required to complete an interrupt-acknowledge sequence have been described in an earlier section.

Physical Design Considerations

When designing a system using the 80286 processor, the physical design considerations include the connection and decoupling of power, ground, and the 80286 CAP pin, and physical provisions for debugging with an oscilloscope, logic analyzer, or Intel Integrated In-Circuit Emulator (I^2ICE).

The 80286 processor is packaged in either a JEDEC-approved 68-pin leadless chip carrier, or in a Pin Grid Array package. Figure 3-72 shows the pin configurations for both the pad/pin view (the underside of the component when mounted on a PC board) and the top view (the top of the component when mounted on a PC board). The footprint of the Pin Grid Array package is identical to that of the socket for the leadless chip carrier, as shown in the figure.

POWER, GROUND, AND CAP CONNECTIONS

As shown in Figure 3-73, the 80286 processor has two power pins and three ground pin connections. The two power pins must be connected to +5 volts; all three ground pins must be connected to 0 volts. A 0.1 μF low-impedance decoupling capacitor should be connected between power and ground. This capacitor should be located as close as possible to the CPU socket.

The power and ground connections from the power supply to the CPU should be as low impedance and inductance as practically possible. Large surge currents are possible, for example, when all 24 address outputs change at once.

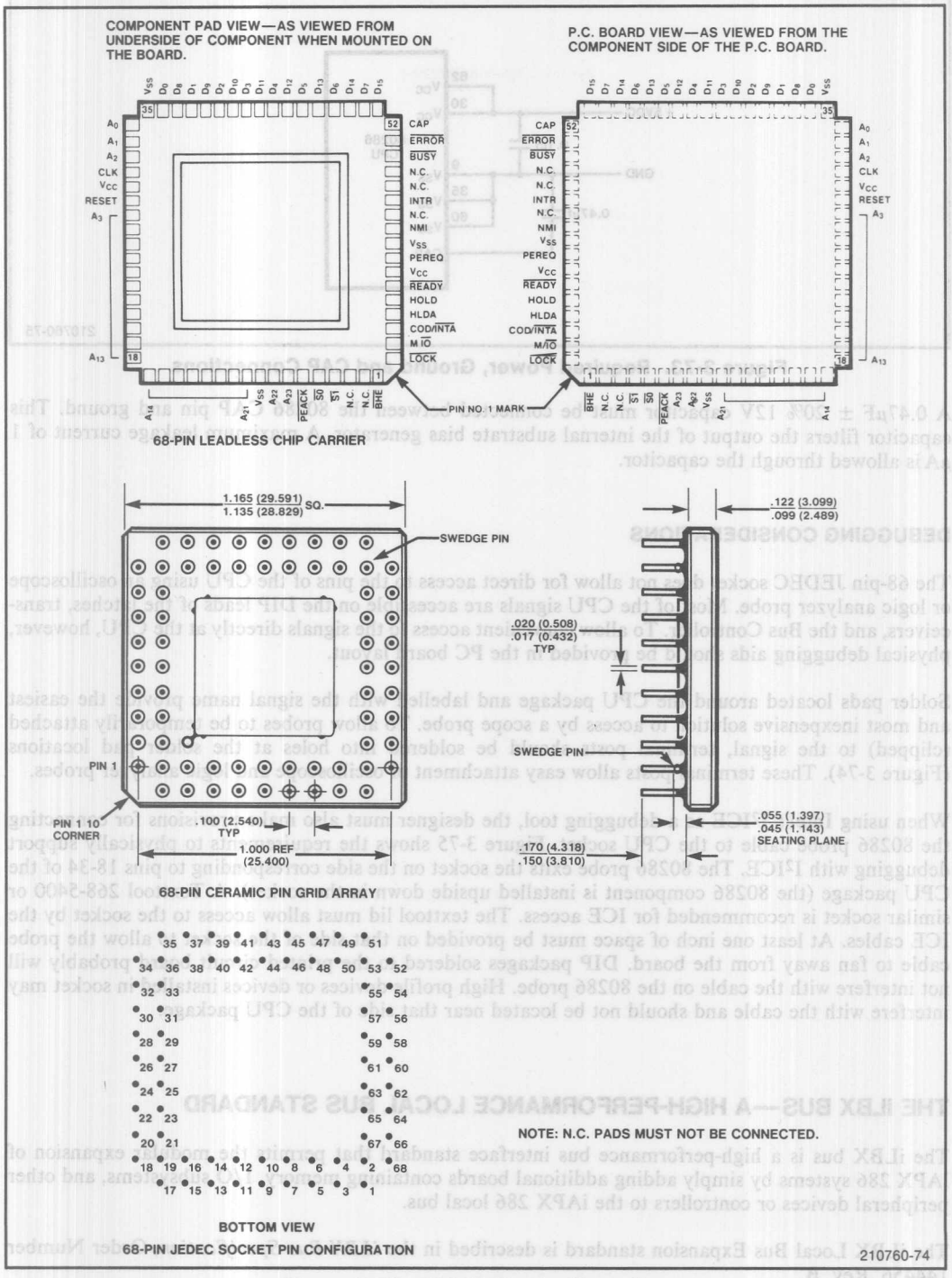


Figure 3-72. 80286 Pin Configuration

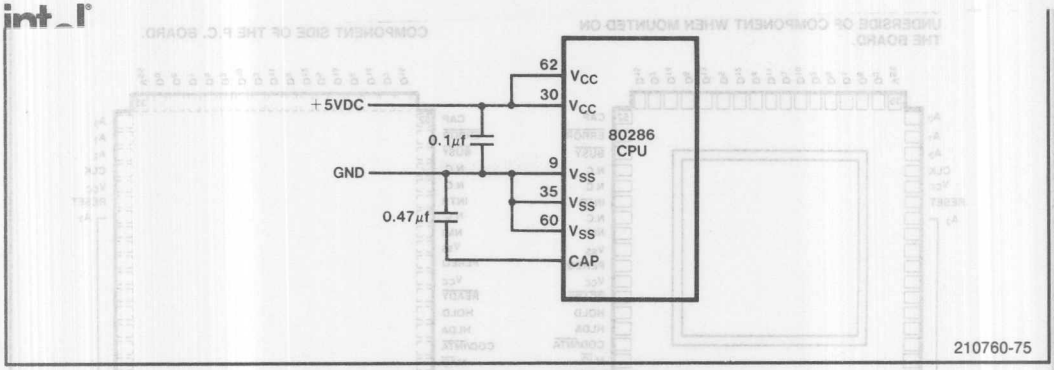


Figure 3-73. Required Power, Ground and CAP Connections

A 0.47μF ± 20% 12V capacitor must be connected between the 80286 CAP pin and ground. This capacitor filters the output of the internal substrate bias generator. A maximum leakage current of 1 μA is allowed through the capacitor.

DEBUGGING CONSIDERATIONS

The 68-pin JEDEC socket does not allow for direct access to the pins of the CPU using an oscilloscope or logic analyzer probe. Most of the CPU signals are accessible on the DIP leads of the latches, transceivers, and the Bus Controller. To allow convenient access to the signals directly at the CPU, however, physical debugging aids should be provided in the PC board layout.

Solder pads located around the CPU package and labelled with the signal name provide the easiest and most inexpensive solution to access by a scope probe. To allow probes to be temporarily attached (clipped) to the signal, terminal posts should be soldered into holes at the solder pad locations (Figure 3-74). These terminal posts allow easy attachment of oscilloscope and logic analyzer probes.

When using Intel's I²ICE as a debugging tool, the designer must also make provisions for connecting the 80286 probe cable to the CPU socket. Figure 3-75 shows the requirements to physically support debugging with I²ICE. The 80286 probe exits the socket on the side corresponding to pins 18-34 of the CPU package (the 80286 component is installed upside down in the socket). A Texttool 268-5400 or similar socket is recommended for ICE access. The texttool lid must allow access to the socket by the ICE cables. At least one inch of space must be provided on that side of the socket to allow the probe cable to fan away from the board. DIP packages soldered to the printed circuit board probably will not interfere with the cable on the 80286 probe. High profile devices or devices installed in socket may interfere with the cable and should not be located near that side of the CPU package.

THE iLBX BUS—A HIGH-PERFORMANCE LOCAL BUS STANDARD

The iLBX bus is a high-performance bus interface standard that permits the modular expansion of iAPX 286 systems by simply adding additional boards containing memory, I/O subsystems, and other peripheral devices or controllers to the iAPX 286 local bus.

The iLBX Local Bus Expansion standard is described in the *iLBX Bus Specification*, Order Number 144456, Rev. B.

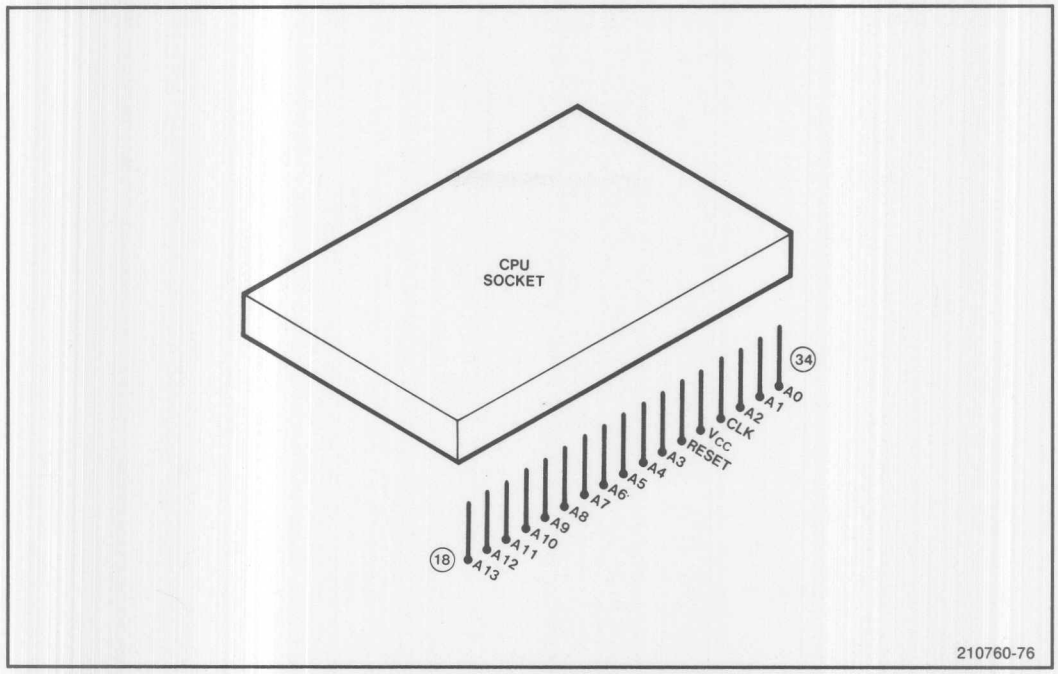


Figure 3-74. Terminal Posts Provide Signal Access

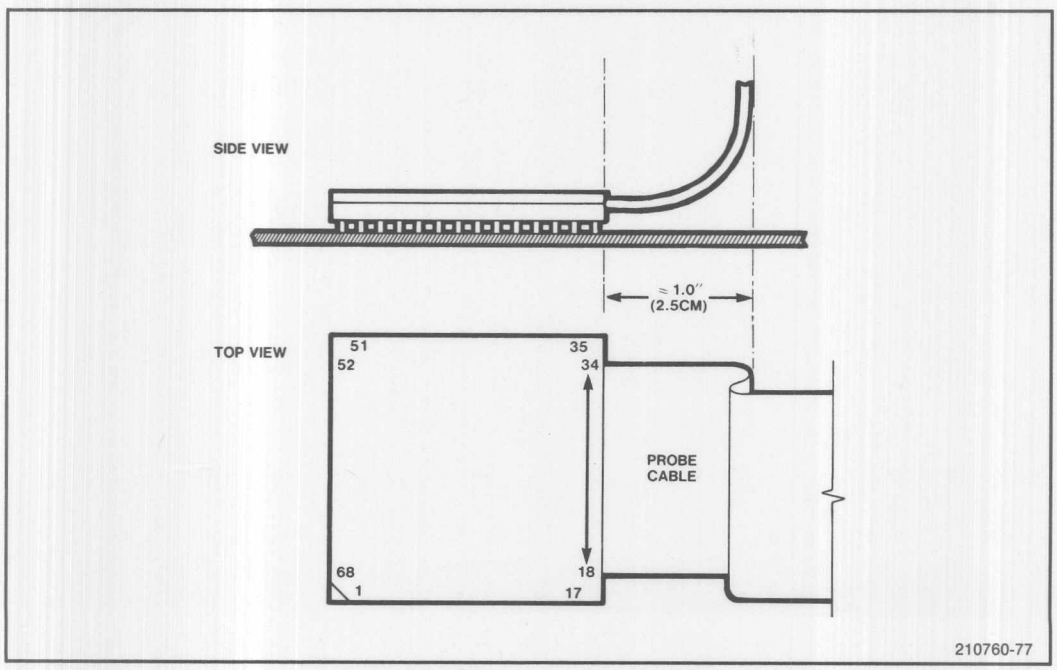


Figure 3-75. I²ICE™ Probe Cabling Requirements

Figure 3-74. Terminal Posts Provide Signal Access

Figure 3-75. iICE™ Probe Cabling Requirements

Memory Interfacing 4

MEMORY INTERFACING

This chapter provides guidelines for designing memory subsystems for the iAPX 286.

One of the principal considerations in designing memory subsystems for the iAPX 286 is the effect of memory performance on the performance of the overall iAPX 286 system. In this chapter, the performance implications of specific memory designs are examined in detail:

- In the first section of this chapter, the tradeoff between system performance and memory cost is examined in detail. The important relationships to consider are the relationship of system performance to memory performance, memory performance to memory device speed, and, finally, memory device speed to memory cost.
- The second section uses benchmark test results to identify the actual relationship between memory performance (measured by the number of required wait states) and overall system performance. The impact of wait states on overall system performance is not as severe as you might initially expect.
- In the third section, several memory interface techniques are introduced to show that memory system performance is determined as much by the memory system design as it is by the actual speed of the memory devices used. Specifically, two interface techniques can be used to gain increased memory performance while using relatively slow memories.
- The fourth section contains a detailed timing analysis of the three memory interface techniques introduced in the previous section.
- The fifth section of this chapter contains specific details for interfacing common types of memory devices to the iAPX 286. ROMs, static RAMs, and pseudo-static RAMs are described, as well as the use of the 8207 Advanced Dynamic RAM Controller to interface to dynamic RAMs.

MEMORY SPEED VS. PERFORMANCE AND COST

In a high-performance microprocessing system, overall system performance is linked very closely with the performance of its memory subsystems. Memory is used for program storage, and for the storage of data and information used in processing. The vast majority of bus operations in a typical microprocessing system are operations to or from memory.

It makes little sense to couple a high-performance CPU with low-performance memory; a high-performance CPU running in a system with a large number of wait states provides no better throughput than a low-performance processor, and is certainly more expensive. To realize the performance potential of the 80286 CPU, it is imperative that the CPU be coupled to relatively fast memory.

At the same time, however, fast memory devices are more expensive than slower memory devices. In a system such as the iAPX 286 supporting up to sixteen megabytes of addressable memory, it is clear that providing a large physical memory space using the fastest available memory devices would result in an exceedingly costly design.

This cost/performance tradeoff can be mediated to some extent by partitioning functions and using a combination of both fast and slow memories. Locating the most frequently-used functions in fast memory and the less-used functions in slower memory will reduce costs over a system that uses fast memory throughout. For example, in a RAM-based system that uses read-only memory devices primarily during

initialization, the PROM/EPROM can be very slow (3-4 wait states) with little affect on system performance. RAM memory can also be partitioned into fast local memory and slower system memory.

It is clear, then, that designers must strike a balance between system performance and system costs, and that the choice of memory subsystems is the balancing factor. In order to intelligently tradeoff cost versus performance, however, it is useful to examine the principle relationships that make up this balance.

Three relationships tie together system cost and system performance:

1. System performance as a function of memory performance
2. Memory performance as a function of memory subsystem design and memory device speed
3. Memory device speed as related to memory subsystem cost

System Performance and Memory Performance

System performance measures the speed at which the microprocessing system performs a given task or set of instructions. Memory performance measures the speed at which the microprocessor can access or store a single item of information into memory. To accommodate slower memories, wait states can be inserted into the 80286 bus cycle. A less-common alternative is to use a slower clock frequency for the 80286, thereby lengthening the 80286 bus cycle.

Each wait state that is inserted into a bus operation represents a 50% increase in bus cycle time over zero-wait-state operation. Although at first glance you might expect memory performance and overall system performance to track each other in a one-to-one relationship, this is not actually the case.

As explained in the following section, the increased bus cycle time for each additional wait state results in an average increase of only 25% in overall CPU execution time, rather than the 50% increase that simple hand calculations might predict. Benchmark tests reveal that overall CPU execution time increases by about 25% over zero-wait operation when running with 1 wait state, and an additional 25% over zero-wait operation for each additional wait state. The following section of this chapter explains the architectural and operational characteristics of the 80286 which reduce the effect of wait states on system performance.

The 80286 clock frequency is directly related to system performance; execution time increases in direct proportion to the increase in clock period, or the reduction in clock frequency. A 6-MHz 80286 requires 33 percent more time to execute a program than an 8 MHz 80286 operating with the same number of wait states. Since a slower clock frequency increases the 80286 bus cycle times, however, fewer wait states may be required to accommodate slow memory devices. A slight reduction in clock frequency may actually increase system performance in some instances if the slower clock allows the same memories to be used with fewer wait states.

Memory Speed and Memory Performance

The relationship between memory subsystem performance and the speed of individual memory devices is determined by the design of the memory subsystem. A later section of this chapter describes several design alternatives that permit relatively high-performance memory subsystems to be designed using relatively slower memory devices. These design alternatives include the use of special address strobe logic and the use of interleaved memory banks.

Special address strobe logic can be used in place of the 82288 ALE signal to generate a valid address to the memory subsystem earlier in the 80286 bus cycle. This technique results in an increased memory access time, permitting the use of slower memories without requiring additional wait states.

Interleaving memory access between two or more banks of memory devices can also increase the memory access time for most bus cycles. Using interleaved memory, memory devices are grouped into banks so that each sequential fetch comes from the next memory bank. In a memory system partitioned into four memory banks, for example, a program that fetches four 16-bit words from consecutive addresses actually fetches one word from each bank. The early address generated by the CPU can be latched and made available to the next memory bank while the address for the previous cycle is still valid for the previously-selected memory bank. These memory interface techniques are described in more detail later in this chapter.

iAPX 286 SYSTEM PERFORMANCE WITH WAIT STATES

The iAPX 286 system supports wait states in the 80286 bus cycle to allow the use of slower memory and peripheral devices. These wait states extend the time required to perform individual bus operations, however, and so directly influence iAPX 286 system performance.

Wait states extend the 80286 bus cycle by adding an additional T_c state for each wait state desired. For an 8-MHz 80286, each wait state adds an additional 125 ns (an increase of 50%) to the minimum bus cycle time of 250 ns. An 80286 CPU running with one wait state executes a bus cycle in 375 ns, compared with the 250 ns required when operating with zero wait states.

Calculating performance degradation from this figure alone, however, will lead to overly-pessimistic results. On average, benchmark tests show that each wait state adds an additional 25% of the zero-wait execution time to the overall execution time of a task. Simpe hand calculations would predict an increase of 50% over the zero-wait execution time for each wait state.

Table 4-1 shows the results of benchmark tests using four Pascal benchmarks and five assembly-language benchmark programs. These benchmarks were executed on an iAPX 286 system operating with from zero to four wait states. The Pascal benchmarks averaged a 23% increase in execution time for each additional wait state, while the assembly-language benchmarks showed an average 20% increase in execution time for each additional wait state.

Explaining the Benchmark Results

As shown in the benchmark results in Table 4-1, the effect of wait states on iAPX 286 performance varies with the nature of the program that is executing and the types of bus cycles being performed. The pipelining within the 80286 CPU sometimes causes idle bus cycles while the CPU is executing prefetched instructions, and no other bus operations are requested. Wait states reduce the number of these idle bus cycles and, on average, do not produce as severe a degradation in performance as simple performance calculations might predict.

The characteristics of an individual program are important in determining processor performance with wait states. For example, when the CPU is executing a program containing many multiplication and division operations, performance is not degraded significantly by wait states. This is because processor performance is limited by processing speed and not by bus throughput.

During program execution, the 80286 Bus Unit will prefetch instructions and fill the instruction and prefetch queues. These prefetch operations occur several clock cycles ahead of execution and occur

Intel Pascal Benchmarks

Benchmark	Performance	0 Wait States	1 Wait State	2 Wait States	3 Wait States	4 Wait States
Queens	Time (s)	7.19	9.224	11.496	14.43	17.2
	Normalized Time	(1.0)	(1.28)	(1.6)	(2.0)	(2.4)
GCD	Time (s)	12.13	14.665	16.87	20.48	24.23
	Normalized Time	(1.0)	(1.21)	(1.39)	(1.69)	(2.0)
Bubble Sort	Time (s)	5.587	7.586	9.74	11.83	14.64
	Normalized Time	(1.0)	(1.36)	(1.74)	(2.12)	(2.62)
Matrix Mult.	Time (s)	6.92	7.607	9.12	11.08	13.33
	Normalized Time	(1.0)	(1.1)	(1.32)	(1.6)	(1.93)
Average Normalized Time:		(1.0)	(1.23)	(1.51)	(1.85)	(2.23)

Intel Assembly-Language Benchmarks

Program	Performance	0 Wait States	1 Wait State	2 Wait States	3 Wait States
PCALL	Time (us)	17.0	21.38	27.50	33.25
	Normalized Time	(1.0)	(1.26)	(1.62)	(1.96)
BSORT	Time (us)	494.0	633.0	778.0	946.0
	Normalized Time	(1.0)	(1.28)	(1.58)	(1.92)
XLAT	Time (us)	415.0	466.0	565.0	698.0
	Normalized Time	(1.0)	(1.12)	(1.36)	(1.68)
XFORM	Time (ms)	285.0	317.0	346.0	385.0
	Normalized Time	(1.0)	(1.12)	(1.22)	(1.35)
INSPECT	Time (ms)	217.0	254.0	289.0	346.0
	Normalized Time	(1.0)	(1.17)	(1.33)	(1.59)
Average Normalized Time:		(1.0)	(1.19)	(1.42)	(1.70)

when the bus would otherwise be idle. Any wait states in these instruction prefetches will not usually delay program execution.

On the other hand, a program that makes extensive use of the bus for manipulating data will show greater degradation in performance when wait states are introduced. Programs that use many data read and write operations are examples of programs that are bus-limited.

Since data read operations are performed in response to an executing instruction, the Execution Unit must wait for the data to be read. Therefore, any delay in reading the data will have a direct impact on system performance. Each wait state will add an additional processor clock cycle to the execution time of the current instruction. Word data read from odd addresses will result in two back-to-back byte read operations, resulting in twice the delay before completion of the read instruction.

Data write operations are performed by the 80286 Bus Unit from information stored in temporary data and address registers. The Execution unit fills these temporary registers and continues executing. Because

the Execution unit is not delayed, isolated data write operations with several wait states will not affect system performance.

Wait states in the write operation may delay subsequent data read operations or instruction prefetches, however. If several back-to-back data write operations are performed, the 80286 Execution unit may have to wait for the Bus unit to finish before performing the next data write. Word writes to odd addresses will result in twice the number of delays as for word writes to even addresses.

Since the nature of the program has a direct impact on the performance of the iAPX 286 with one or more wait states, any estimate of CPU performance must take this information into account. The best estimates can be made by using the results tabulated for a benchmark program with characteristics most closely matching those of the intended application. The only way to obtain a precise figure for a particular application is to run that application and measure its performance.

The Intel Pascal Benchmarks

The Pascal benchmark results shown in Table 4-1 reflect the performance of an 80286 operating in Real-Address mode. These benchmarks were written in Pascal and compiled using Intel's Pascal-86 compiler version 2.0. The benchmarks themselves are described in the *iAPX 86 System Benchmark Report*, Order Number 210352. The following paragraphs give a brief description of the Intel Pascal benchmarks.

QUEENS (Chess Simulation)

The Intel Queens benchmark lists all possible combinations of non-attacking queens on a 9 × 9 chessboard. This program tests control structures and boolean evaluations, and evaluates the code generated for certain commonly-used statements (A = A + 1, for example).

GCD (Greatest Common Denominator)

This Intel program computes the greatest common denominator of two integers using a recursive function call. Function overhead and the MOD operator are tested by this benchmark.

BUBBLE SORT

This Intel benchmark sorts 1000 integers into numerically-ascending order using the exchange (bubble) sort algorithm. This benchmark extensively tests control structures, relational expressions, and array references.

MATRIX MULTIPLY

This Intel benchmark uses a simple row/column inner product method to compute the product of two 32 × 32 matrices. The elements of the matrices are integers. The benchmark tests control structure arrays, array references, and integer arithmetic.

The Intel Assembly-Language Benchmarks

The assembly-language benchmark results shown in Table 4-1 reflect the performance of an 80286 operating in Protected mode. These benchmarks are described in the *8086 16-Bit Microprocessor Benchmark Report*. The following paragraphs give a brief description of each of these assembly-language benchmarks.

INSPECT (Automated Parts Inspection)

The automated parts inspection program controls an image-dissection camera having two 8-bit D/A converters (for X and Y axis control) and a 12-bit A/D converter that generates a gray-scale signal. For each of the 16,384 (128 × 128) points, the measured gray-scale signal is compared with a known good gray-scale value (stored in memory) to determine if it is within tolerance. One 16-bit multiply and one 16-bit divide are performed for each of the 16,384 points.

XLAT (Block Translation)

The block translation benchmark translates each EBCDIC character from a memory buffer into an ASCII character to be stored in a second buffer. The translation detects when an EOT character is encountered or when the entire EBCDIC buffer has been translated. For the benchmark tests, the EBCDIC buffer contains 132 EBCDIC characters, none of which is an EOT character.

BSORT (Bubble Sort)

The bubble sort benchmark sorts a one-dimensional array containing 16-bit integer elements into numerically-ascending order using the exchange (bubble) sort algorithm. For the benchmark tests, the array contains 10 integers that are initially arranged in descending order.

XFORM (Graphics X-Y Transformation)

The X-Y transformation benchmark expands or compresses (scales) a selected graphics window containing 16-bit unsigned integer X-Y pairs. Each X data value is offset by X_0 and multiplied by a fractional scale factor, while each Y data value is offset by Y_0 and multiplied by the same scale factor. One 16-bit multiply and one 16-bit divide are performed for each of the X-Y coordinates. For the benchmark test, the selected window contains 16,384 X-Y pairs.

PCALL (Reentrant Procedure)

The reentrant-procedure benchmark exercises processor features that are used to implement a reentrant procedure. Three input parameters are passed by value to the reentrant procedure. Prior to the call, one parameter is in ne of the general registers, while the other parameters are stored in memory locations. Upon entry, the reentrant procedure preserves the state of the processor, assuming that it will use all of the general registers. The reentrant procedure then allocates storage for three local variables, adds the three passed parameters, and stores the result in a local variable. Upon exit, the state of the processor is restored.

MEMORY INTERFACE TECHNIQUES

In the previous section, the impact of memory performance and wait states on system performance was examined in detail. In this section, memory performance itself is examined in order to understand how the performance of a memory subsystem is determined as much by the subsystem design as by the

speed of the individual memory devices. Three different memory interface techniques are introduced, and the advantages and tradeoffs of each are described. The *Memory Design Handbook* describes still other techniques that may be used.

Standard Address Strobe Logic

Chapter Three describes a straightforward method for interfacing to memory, using the ALE output of the 82288 Bus Controller to strobe the address and chip-select latches. Figure 4-1 shows this memory-interface model, showing the ALE signal from the 82288 Bus Controller being used to control the 8282 address latches.

The timing of this straightforward interface technique using the standard ALE address strobe is examined in Chapter Three. Although this technique is simple and requires only a small number of devices to implement, it provides only 87.5 ns of address access time at zero-wait-states. This limited access time is sufficient only for very fast static RAM devices; slower memory devices will require additional wait states and corresponding reductions in system performance.

Other, more complex, methods of interfacing memory devices to the 80286 can increase address access time to permit high-performance operation with slower memory devices. The following two sections introduce these two techniques.

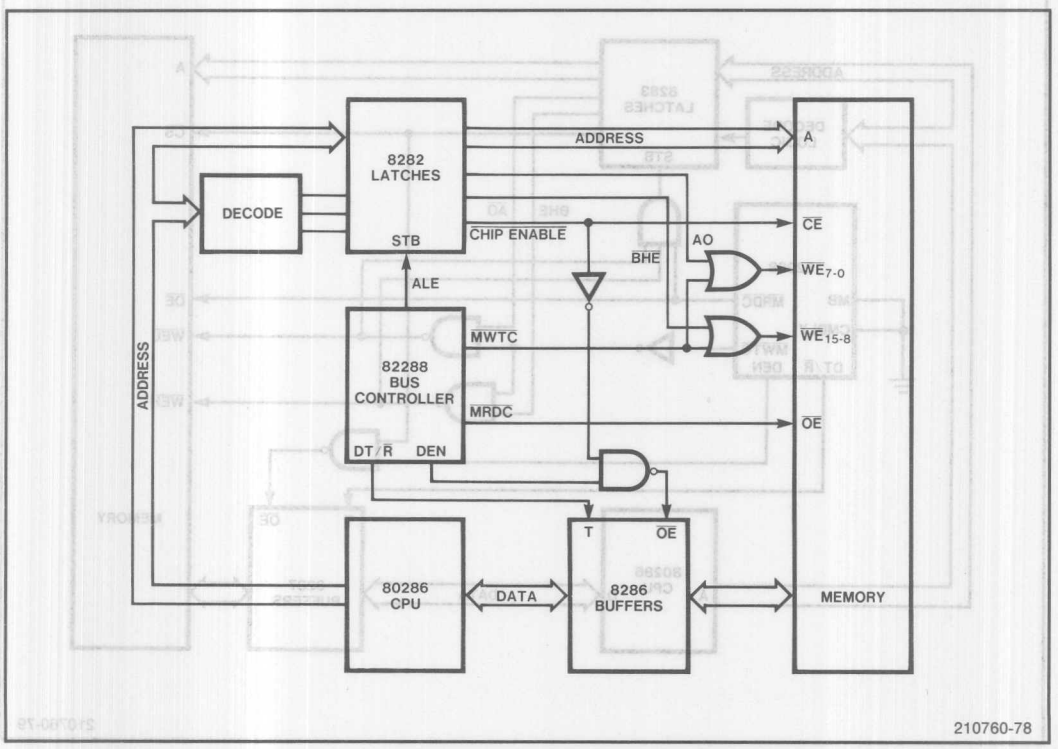


Figure 4-1. Memory Interface Using Standard ALE Signal

As an enhancement to the timing characteristics of the standard memory-interface technique described above, special address strobe logic can be used to generate a valid address to the memory devices at an earlier point in the bus cycle. This special address strobe logic is shown in Figure 4-2.

The address strobe logic shown in Figure 4-2 generates an address strobe as early as possible in the memory cycle; that is, as soon as the command from the previous memory operation becomes inactive. This memory-interface technique trades off address hold time following command inactive (few memory devices require a lengthy address hold) in order to gain additional address access time.

Two additional optimizations are the use of 8283 (inverting) address latches and 8287 (inverting) data transceivers to drive the buffered local bus containing the memory subsystem; these inverting buffers exhibit faster timing characteristics than their non-inverting 8282 and 8286 counterparts. Because these devices invert the data and address lines, however, these latches and transceivers should not be used with I/O devices on the same buffered local bus, to avoid confusion. The reduced address hold time resulting from this special strobe logic also is not compatible with many typical I/O devices.

The timing of this special address strobe logic is described in detail in a later portion of this chapter. The principle advantages of the special strobe logic depicted in Figure 4-2 is that address access time is increased from 87.5 ns at zero wait states using the standard ALE strobe to as much as 155 ns using

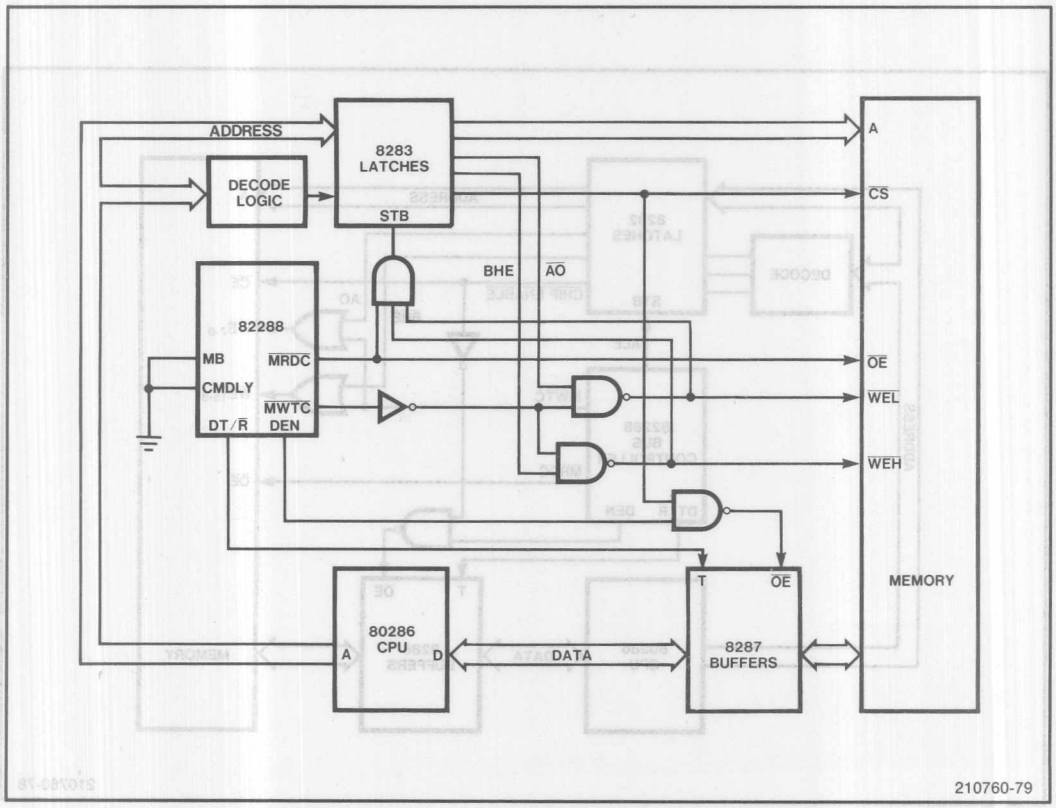


Figure 4-2. Memory Interface Using Special Strobe Logic

the special address strobe. This increased address access time permits slower memory devices to be used in iAPX 286 systems, while still permitting operation at zero wait states.

The tradeoff made when using this special strobe logic is the increased hardware overhead required to implement the strobe logic, and the separate latches and data buffers required for the separate memory and I/O buses.

For systems that do not require large amounts of memory, however, this special strobe logic can provide an economical, high-performance solution. The cost savings of using slower memories to obtain a given level of performance more than offsets the cost of the additional logic required.

Interleaved Memory

For systems that require large amounts of memory, an interleaved memory subsystem can provide even greater economies than using the special strobe logic just described, at the cost of extra address latches and additional control logic.

With an interleaved memory subsystem, memory devices are grouped into banks so that every sequential memory fetch comes from a different memory bank. For example, in a memory system partitioned into four memory banks, a program that fetches four 16-bit words from consecutive addresses actually fetches one word from each bank.

Interleaved memory banks permit designers to take full advantage of the 80286's pipelined address timing. By letting the early address from the 80286 propagate into the next memory bank while the address for the current memory cycle is still valid in another bank, address access times for the individual memory devices can be greatly increased.

Of course, occasional back-to-back memory cycles to the same memory bank may occur, which will require a longer bus cycle than if the access was to a different bank. The interleave control logic must accommodate the longer bus cycle times by inserting an additional wait state into the cycle. On the average, though, these memory "hits" comprise only 7% of all memory cycles using two interleaved memory banks, and 80286 execution time for typical software is increased only 4% on average over execution exclusively at zero wait states.

Figure 4-3 shows an example circuit that uses interleaved addressing and special address strobes to increase the access time for most memory cycles. The memory devices are configured into two banks with successive memory words assigned to alternate banks (address line 1 is used as a bank select). Each bank has its own address latches and strobe signal. An 8205 decoder decodes the CPU address to generate chip enables for the individual memory devices, and a $\overline{\text{BANKSEL}}$ signal is decoded to enable the address strobes.

Figure 4-4 shows the logic that generates the address strobe signals to the individual banks, and detects consecutive reads to the same bank. Generally, the address strobes will be high until a read to one of the memory banks occurs. The high address strobes maintain the address latches in their transparent state to enable an address generated by the CPU to be transmitted to the memories in the shortest possible time. (With the standard memory model, the address would not be transmitted to memory until just before the start of T_c due to the 82288 ALE timing and strobe to output delays.)

The address strobe to the selected bank is driven low on the falling edge of CLK at the start of the first T_c state to latch the address and chip selects to the respective memory devices. The address strobe to the non-selected bank remains high, maintaining the respective address latches in their transparent state.

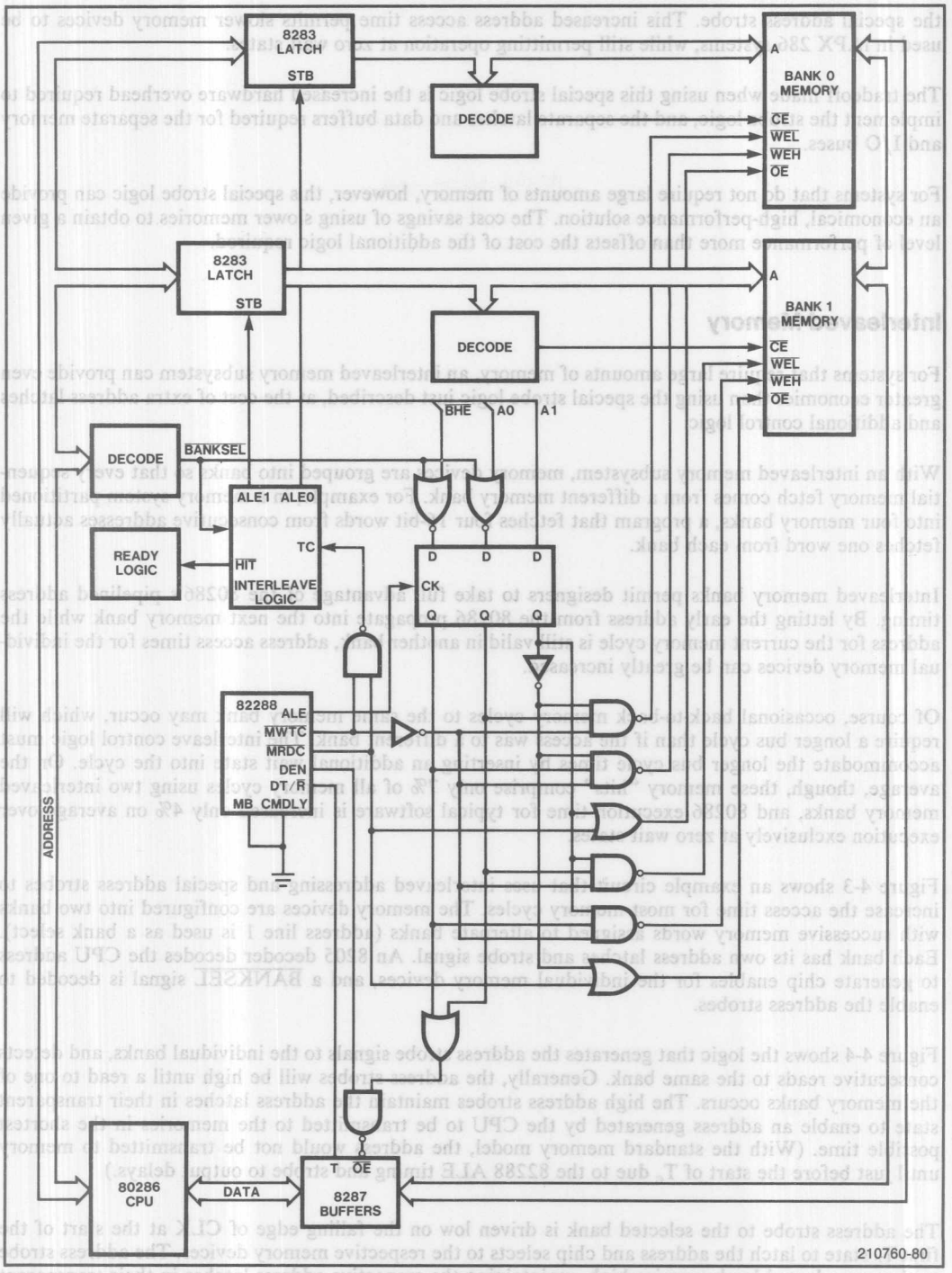


Figure 4-3. Interleaved Memory Circuit

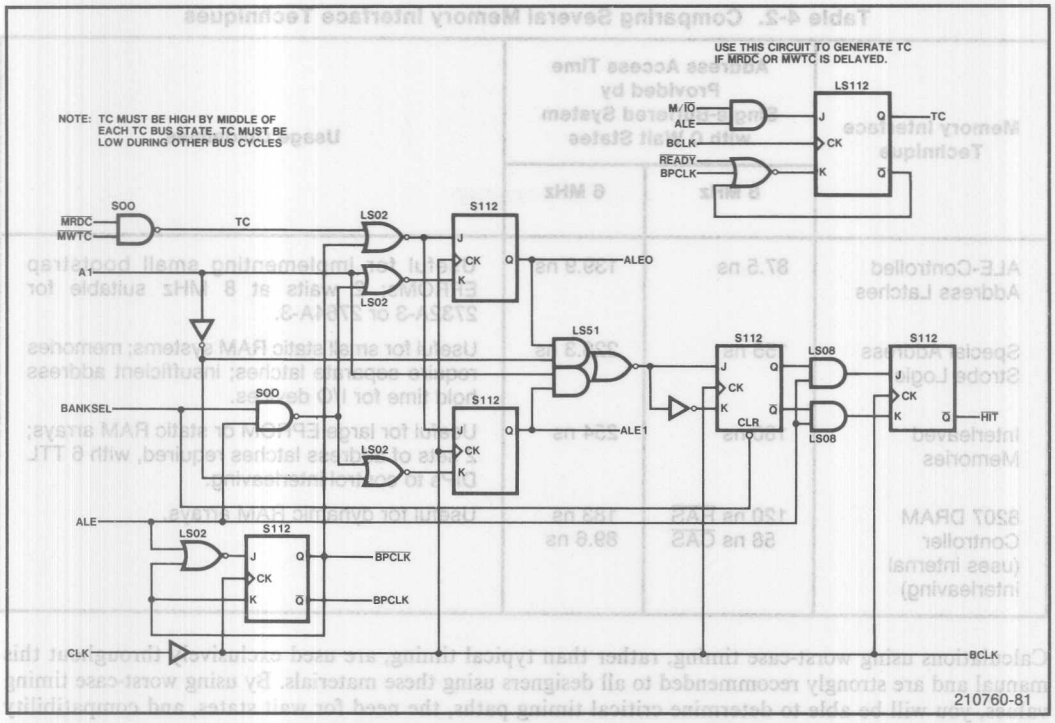


Figure 4-4. Address Strobe Generation

The timing of this interleaved memory system is described in a later section of this chapter. The principal advantage of using an interleaved memory system such as this is the increased access time provided at the beginning of most 80286 bus cycles. Access time requirements as long as 180 ns can be accommodated using this configuration, while still permitting most 80286 memory operations to occur with zero wait states. With one wait state, access times as long as 305 ns can be reliably accommodated, as compared with access times of 212.5 ns for the standard memory configuration at one wait state.

This increased access time results in a direct cost savings without reducing performance. For an example system using EPROMs with one wait state, the savings from interleaving are composed of the savings from using 300 ns EPROMS instead of the more expensive 200 ns EPROMS, multiplied by the number of EPROMS, and subtracting the additional interleave circuitry. For large-memory systems, this cost savings without reducing performance can be well worth the effort.

Table 4-2 compares the various memory interface techniques and provides guidelines on selecting a particular technique for specific applications. The following section analyzes the timing of each of these interface techniques in more detail.

TIMING ANALYSIS FOR MEMORY OPERATIONS

To design a memory subsystem for the iAPX 286, designers must understand the timing of the 80286 memory cycle. In the sections that follow, critical timing values are calculated for each of the memory interface techniques described in the previous section. Timing calculations are shown primarily for an 8-MHz 80286; in some cases, timing values for a 6-MHz 80286 are included to give examples of these calculations as well.

Memory Interface Technique	Address Access Time Provided by Single-Buffered System with 0 Wait States		Usage Guidelines
	8 MHz	6 MHz	
ALE-Controlled Address Latches	87.5 ns	139.9 ns	Useful for implementing small bootstrap EPROMs; 2 waits at 8 MHz suitable for 2732A-3 or 2764A-3.
Special Address Strobe Logic	155 ns	228.3 ns	Useful for small static RAM systems; memories require separate latches; insufficient address hold time for I/O devices.
Interleaved Memories	180 ns	254 ns	Useful for large EPROM or static RAM arrays; 2 sets of address latches required, with 6 TTL DIPs to control interleaving.
8207 DRAM Controller (uses internal interleaving)	120 ns $\overline{RAS}$ 58 ns $\overline{CAS}$	183 ns 89.6 ns	Useful for dynamic RAM arrays.

Calculations using worst-case timing, rather than typical timing, are used exclusively throughout this manual and are strongly recommended to all designers using these materials. By using worst-case timing values, you will be able to determine critical timing paths, the need for wait states, and compatibility between various memory devices.

Analyses using typical timing parameters make implicit assumptions about the system under study. Typical calculations assume that with 5 to 8 devices in series, the probability of all devices simultaneously exhibiting worst-case timing is acceptably remote. However, since many modern designs use only 2 or 3 devices in series, the probability of all devices showing worst-case timing characteristics becomes unacceptably high.

For this reason, worst-case timing calculations are used to assure reliable operation over all variations in temperature, voltage, and individual device characteristics. Worst-case timing values are determined by assuming the maximum delay in the latched address, chip-select, and command signals, and the longest propagation delays through data buffers and transceivers.

Standard ALE Timing

The iAPX 286 memory cycle using the standard ALE timing from the 82288 Bus Controller is described in Chapter Three. The analysis of timing is largely dependent on the particular memory configuration. Figure 4-1, shown previously, shows the memory model used to analyze timing using the standard ALE signal timing. Address and chip selects are measured from the outputs of latches (strobed by ALE). Data valid is measured at the data pins of the CPU. $\overline{MRDC}$ is an 82288 output.

A single data buffer is used to prevent the AC loading from the memory subsystem from exceeding the 80286 capacitive limit of 100 pF. The 8286 buffers can drive 300 pF loads. For very small systems, the 80286 can be used without data buffers. For large, multi-board systems, two levels of buffering may be used. In any case, the appropriate buffer delays should be considered when performing timing calculations.

READ OPERATIONS

Figure 4-5 shows the timing for zero-wait state memory read cycles using the standard ALE address strobe.

For read operations, data must be valid at the pins of the 80286 10 ns before the falling edge of CLK at the end of the final T_c (20 ns for the 6-MHz 80286). The critical timing parameters that determine whether one or more wait states are required for a given memory subsystem design are the required address and command access times before read data will be valid.

The maximum allowable address and chip-enable access times for the memory configuration shown in Figure 4-1, with zero wait states, and incorporating one transceiver delay, can be calculated as follows:

	8 MHz	6 MHz
3 CLK cycles	187.5 ns	249.9 ns
− ALE active delay (max)	− 15.0 ns	− 15.0 ns
− 8282 stb-to-output delay (max)	− 45.0 ns	− 45.0 ns
− 8286 transceiver delay (max)	− 30.0 ns	− 30.0 ns
− 80286 required data setup (min)	− 10.0 ns	− 20.0 ns
Maximum address to output delay =	87.5 ns	139.9 ns

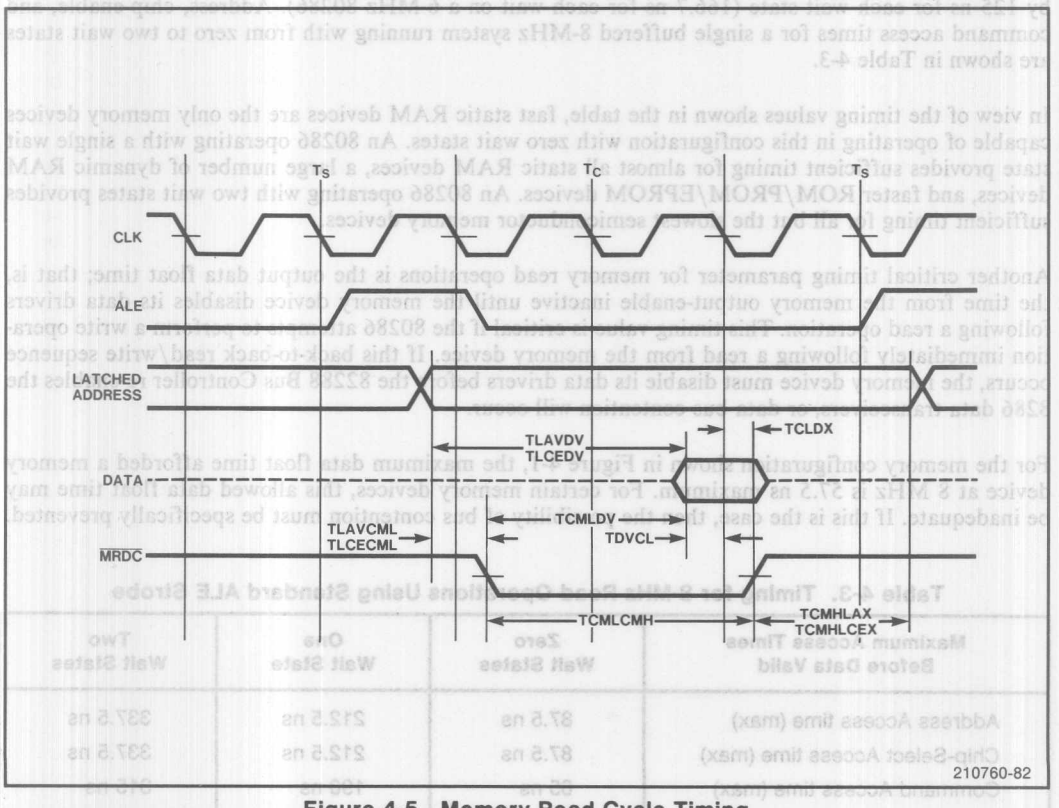


Figure 4-5. Memory Read Cycle Timing

The calculations for maximum allowed command or output-enable access time before data valid are similar:

2 CLK cycles at 8 MHz	125 ns
− MRDC active delay (max)	− 20 ns
− 8286 transceiver delay (max)	− 30 ns
− 80286 required data setup (min)	− 10 ns
Maximum command to output delay =	65 ns max.

Because of the pipelined address timing of the 80286 CPU, the memory configuration shown in Figure 4-1 provides a significant timing interval after the 80286 address is valid and before the address latch (ALE) strobe becomes active. This interval can be used for address decoding without reducing the chip-enable access times calculated above. The maximum address decode time that this memory configuration can provide without reducing the chip-enable access time is calculated as follows:

2 CLK cycles at 8 MHz	125 ns
− 80286 address valid delay (max)	− 60 ns
+ ALE active delay (min)	+ 3 ns
Maximum decode time =	68 ns max.

This maximum decode time is not affected by the use of wait states in the 80286 bus cycle.

If wait states are inserted into the memory cycle, access times for the relevant parameters are increased by 125 ns for each wait state (166.7 ns for each wait on a 6-MHz 80286). Address, chip-enable, and command access times for a single buffered 8-MHz system running with from zero to two wait states are shown in Table 4-3.

In view of the timing values shown in the table, fast static RAM devices are the only memory devices capable of operating in this configuration with zero wait states. An 80286 operating with a single wait state provides sufficient timing for almost all static RAM devices, a large number of dynamic RAM devices, and faster ROM/PROM/EPROM devices. An 80286 operating with two wait states provides sufficient timing for all but the slowest semiconductor memory devices.

Another critical timing parameter for memory read operations is the output data float time; that is, the time from the memory output-enable inactive until the memory device disables its data drivers following a read operation. This timing value is critical if the 80286 attempts to perform a write operation immediately following a read from the memory device. If this back-to-back read/write sequence occurs, the memory device must disable its data drivers before the 82288 Bus Controller re-enables the 8286 data transceivers, or data bus contention will occur.

For the memory configuration shown in Figure 4-1, the maximum data float time afforded a memory device at 8 MHz is 57.5 ns maximum. For certain memory devices, this allowed data float time may be inadequate. If this is the case, then the possibility of bus contention must be specifically prevented.

Table 4-3. Timing for 8-MHz Read Operations Using Standard ALE Strobe

Maximum Access Times Before Data Valid	Zero Wait States	One Wait State	Two Wait States
Address Access time (max)	87.5 ns	212.5 ns	337.5 ns
Chip-Select Access time (max)	87.5 ns	212.5 ns	337.5 ns
Command Access time (max)	65 ns	190 ns	315 ns

To accommodate the data float time requirements of slower memory and I/O devices, the output enables of the data bus transceivers must be delayed for the necessary length of time. Figure 4-6 shows an example circuit that delays enabling the data transceivers following a read operation to the selected memory devices. This circuit provides a maximum data float time for memory devices of:

2 CLK cycles at 8 MHz (Cmd inact.—DEN act.)	125 ns
− Cmd inactive delay (max)	− 15 ns
+ 8286 enable time (min)	+ 10 ns
Maximum data float time (using circuit) =	120 ns max.

WRITE OPERATIONS

For typical write operations to a memory device, data is latched into the device on the rising edge of the Write command at the end of T_c. The important timing parameters determining whether wait states are required are the address and command access times before command high.

The timing for memory write cycles, shown in Figure 4-7, uses the same model as that for read cycles. Address, chip select/enable, and command generation are the same as for read operations.

Since the latest time that data is active coincides with the latest possible latched address and chip select times (2.5 ns before the falling edge of CLK at the start of T_c), the maximum address-valid to command-high access time provided during write operations with zero wait states is:

3 CLK cycles at 8 MHz	187.5 ns
− ALE active delay (max)	− 30.0 ns
− 8282 address latch delay (max)	− 30.0 ns
+ Cmd inactive delay (min)	+ 3.0 ns
Maximum address access time =	130.5 ns max.

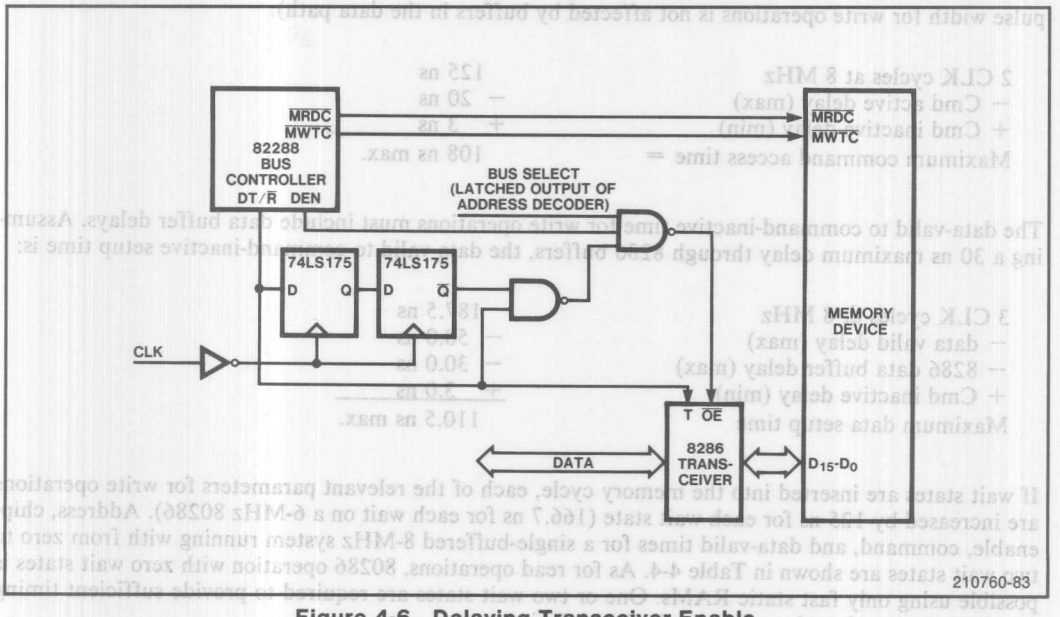


Figure 4-6. Delaying Transceiver Enable

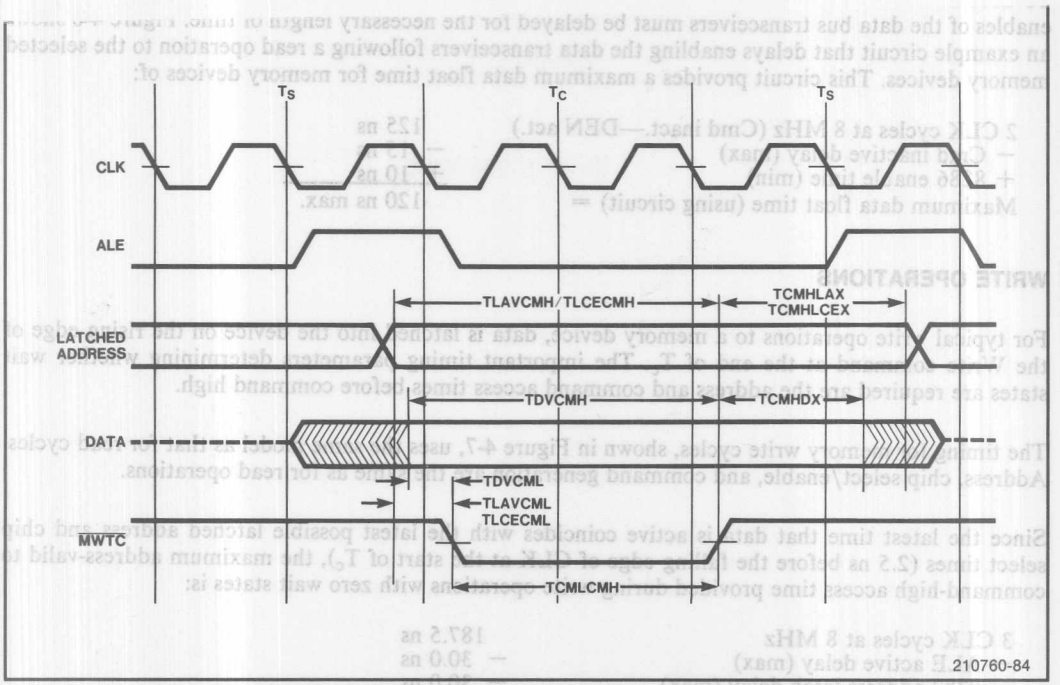


Figure 4-7. Memory Write Cycle Timing Using Standard ALE Strobe

Worst-case command-active to command-inactive timing is calculated in a similar manner (the command pulse width for write operations is not affected by buffers in the data path):

2 CLK cycles at 8 MHz	125 ns
− Cmd active delay (max)	− 20 ns
+ Cmd inactive delay (min)	+ 3 ns
Maximum command access time =	108 ns max.

The data-valid to command-inactive time for write operations must include data buffer delays. Assuming a 30 ns maximum delay through 8286 buffers, the data-valid to command-inactive setup time is:

3 CLK cycles at 8 MHz	187.5 ns
− data valid delay (max)	− 50.0 ns
− 8286 data buffer delay (max)	− 30.0 ns
+ Cmd inactive delay (min)	+ 3.0 ns
Maximum data setup time	110.5 ns max.

If wait states are inserted into the memory cycle, each of the relevant parameters for write operations are increased by 125 ns for each wait state (166.7 ns for each wait on a 6-MHz 80286). Address, chip-enable, command, and data-valid times for a single-buffered 8-MHz system running with from zero to two wait states are shown in Table 4-4. As for read operations, 80286 operation with zero wait states is possible using only fast static RAMs. One or two wait states are required to provide sufficient timing for the majority of static and dynamic RAM devices.

Table 4-4. Timing for 8-MHz Write Operations Using Standard ALE Strobe

Maximum Access Times Before Command High	Zero Wait States	One Wait State	Two Wait States
Address Access time (max)	130.5 ns	255.5 ns	380.5 ns
Chip-Select Access time (max)	130.5 ns	255.5 ns	380.5 ns
Command Pulse width (max)	108.0 ns	233.0 ns	358.0 ns
Write Data Setup Time (max)	110.5 ns	235.5 ns	360.5 ns

The CMDLY and CEN inputs to the 82288 can significantly alter bus cycle timing. CMDLY can delay commands to produce more address, chip enable, and data setup time before a command is issued. CEN can hold commands and data buffer control signals inactive, also altering bus cycle timing. When used, the effects of these inputs must also be included in any worst-case timing analysis.

For some RAM devices, the data setup time before write-command active is a critical parameter. With no command delays, the write command may become active before the write data is valid at the pins of the memory device:

1 CLK cycle at 8 MHz	62.5 ns
− data valid delay (max)	− 50.0 ns
− 8286 data buffer delay (max)	− 30.0 ns
+ Cmd active delay (min)	+ 3.0 ns
Maximum data setup time	− 14.5 ns max.

(Note that the negative setup time implies that data may not be valid before the write command is asserted).

With one command delay, the write command from the 82288 Bus Controller is delayed by 62.5 ns, producing an acceptable 48 ns data setup time before command active. This command delay reduces the command pulse width by 62.5 ns, however, and so additional wait states may be necessary to meet this and other timing requirements.

If the circuit shown in Figure 4-6 is used to delay transceiver enable during write commands, the data setup time is affected again. Using this circuit to accommodate slow output-data-float times by delaying DEN results in a data setup time to end of command of 56 ns minimum:

2 CLK cycles at 8 MHz (DEN-to-WR)	125 ns
− S04 inverter (CLK) delay (max)	− 7 ns
− S175 Flip-flop delay (max)	− 20 ns
− S00, S30 gate delays (max)	− 15 ns
− 8286 output enable time (max)	− 30 ns
+ Cmd active delay (min)	+ 3 ns
Minimum data setup time =	56 ns min.

At the end of the write operation, some RAM devices require that the write data remain valid for a short interval following write command inactive. The maximum data hold requirement permitted by the 80286 is:

1 CLK cycle at 8 MHz (after Cmd)	62.5 ns
− 82288 Cmd inactive delay (max)	− 15.0 ns
+ 8286 Transceiver delay (min)	+ 5.0 ns
Maximum data hold time =	52.5 ns max.

Some memory devices require that the address remain valid following a write operation. For this standard memory configuration, the maximum address hold time following write is:

1 CLK cycle at 8 MHz (after Cmd)	62.5 ns
− 8288 Cmd inactive delay (max)	− 15.0 ns
+ ALE active delay (min)	+ 3.0 ns
+ 8282 STB-to output delay (min)	+ 10.0 ns
Maximum address hold time =	60.5 ns max.

Most memory devices do not require such a lengthy address hold time following a write operation. The following section shows how the special strobe logic introduced previously trades away some of this unused address hold time to gain increased address access time.

Special Address Strobe Timing

As described in a previous section, special address strobe logic can be used to generate an address latch strobe much earlier in the 80286 bus cycle than the 82288 ALE strobe. This special address strobe technique trades off address hold time after command inactive for additional address and chip-enable access time, permitting slow memories to be used with fewer wait states relative to the standard address strobe technique described in the previous section.

Figure 4-2 shown previously shows the memory configuration that will be used in this analysis. Figure 4-8 shows the timing of this special address strobe logic.

The timing of this address strobe logic is relatively straightforward. Maximum address and chip-select access time for an 80286 operating at zero wait states is:

	8 MHz	6 MHz
4 CLK cycles	250 ns	333.3 ns
− Cmd inactive time (max)	− 15 ns	− 15.0 ns
− STB logic delay (max)	− 8 ns	− 8.0 ns
− 8283 latch delay (max)	− 40 ns	− 40.0 ns
− 8287 transceiver delay (max)	− 22 ns	− 22.0 ns
− 80286 required data setup (min)	− 10 ns	− 20.0 ns
Maximum address access time =	155 ns	228.3 ns

Minimum address hold after command inactive is only 10 ns minimum, equal to the minimum 8283 STB-to-output delay.

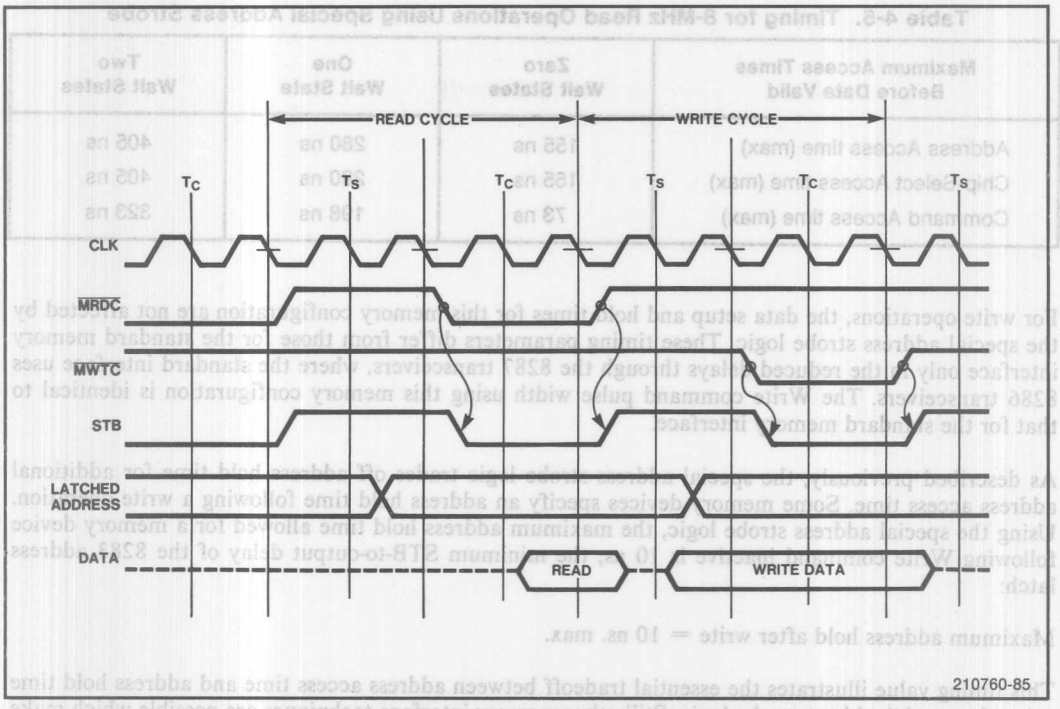


Figure 4-8. Timing for Special Address Strobe Logic

Since the memory configuration using the special strobe logic does not alter the command inputs to the memory devices, command access times are the same as for the standard memory interface, but for the reduced delays through the 8287 transceivers:

2 CLK cycles at 8 MHz	125 ns
− MRDC active delay (max)	− 20 ns
− 8287 transceiver delay (max)	− 22 ns
− 80286 required data setup (min)	− 10 ns
Maximum command to output delay =	73 ns max.

Because the address strobes are generated earlier in the bus cycle, the allowed address decode time before the address strobe goes high is reduced. Maximum decode time without reducing chip-enable access time is:

1 CLK cycle at 8 MHz	62.5 ns
− 80286 address delay (max)	− 60.0 ns
+ Cmd inact. delay (max)	+ 15.0 ns
+ STB logic delay (max)	+ 8.0 ns
Maximum address decode time =	25.5 ns max.

If wait states are inserted into the memory cycle, access times for the relevant parameters are increased by 125 ns for each wait state (166.7 ns for each wait state on a 6-MHz 80286). Address, chip-enable, and command access times for a single-buffered 8-MHz system running with from zero to two wait states are shown in Table 4-5.

Maximum Access Times Before Data Valid	Zero Wait States	One Wait State	Two Wait States
Address Access time (max)	155 ns	280 ns	405 ns
Chip-Select Access time (max)	155 ns	280 ns	405 ns
Command Access time (max)	73 ns	198 ns	323 ns

For write operations, the data setup and hold times for this memory configuration are not affected by the special address strobe logic. These timing parameters differ from those for the standard memory interface only in the reduced delays through the 8287 transceivers, where the standard interface uses 8286 transceivers. The Write command pulse width using this memory configuration is identical to that for the standard memory interface.

As described previously, the special address strobe logic trades off address hold time for additional address access time. Some memory devices specify an address hold time following a write operation. Using the special address strobe logic, the maximum address hold time allowed for a memory device following Write command inactive is 10 ns, the minimum STB-to-output delay of the 8283 address latch:

Maximum address hold after write = 10 ns. max.

This timing value illustrates the essential tradeoff between address access time and address hold time using the special address strobe logic. Still other memory interface techniques are possible which make other allowances in order to optimize the critical address access parameter.

Interleaved Memory Timing

Interleaved memory subsystems have been introduced in a previous section. The interleaving technique, like the special strobe logic, produces increased address access times and allows considerably slower memory devices to be used while still providing performance equal to that of faster memories using the standard memory interface.

An example circuit showing an interleaved RAM system is shown in Figures 4-3 and 4-4. The timing for this circuit is shown in Figure 4-9. This figure shows a sequence of four CPU bus cycles performed in the following order:

1. Write cycle to non-interleaved memory
2. Read cycle from RAM bank 1
3. Read cycle from RAM bank 2
4. Read cycle from RAM bank 2

The sequence of signals required to perform these four bus cycles are as follows:

1. During the write cycle to RAM, both address strobes (ALE0 and ALE1) are high. Since the interleaved memory is not selected, chip enables to the interleaved memories are not generated.
2. At the start of the read cycle from RAM bank 1, the address decode logic generates chip-enable signals to the appropriate devices and $\overline{\text{BANKSEL}}$ activates the interleave logic to generate an appropriate address strobe. ALE1 toggles low on the falling edge of CLK at the end of T_s to latch

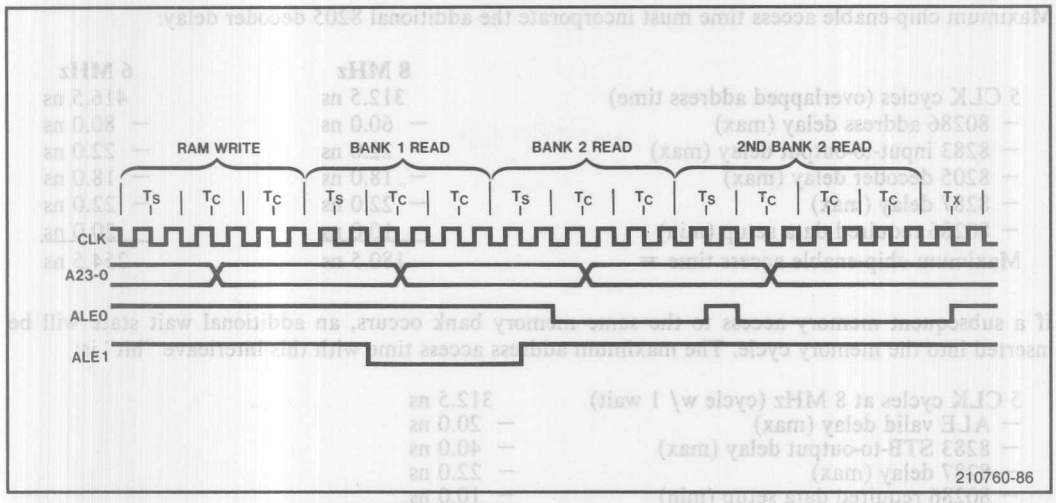


Figure 4-9. Interleaved Memory Timing

the address and chip-enable signals to bank 1. $\overline{\text{MRDC}}$ enables the output drivers of the RAMs. Note that ALE0 remains high throughout the bus cycle, maintaining the Bank 2 address latches in their transparent states.

3. At the start of the read cycle from RAM bank 2, the address decode logic generates chip-enable signals to the appropriate devices as before. ALE1 is driven high on the falling edge of CLK halfway through T_s, and remains high throughout the rest of the bus cycle. ALE0 toggles low at the start of T_c to latch the address and chip-enable signals to bank 2. $\overline{\text{MRDC}}$ enables the output drivers of the RAMs in bank 2, while DEN and the latched $\overline{\text{BANKSEL}}$ enable the interleave-memory data transceivers.

4. The second read cycle to EPROM bank 2 begins with ALE0 low. ALE0 is not driven high until CLK goes low halfway through T_s. This delay reduces the data access time in comparison to the previous bus cycles; the delay is accommodated by adding one additional wait state to the bus cycle. Two S112 flip-flops are used to compare the bank-selects for the current and previous memory cycles to detect the occurrence of back-to-back cycles to the same memory bank. The $\overline{\text{HIT}}$ output of this interleave logic is used to drive the system Ready logic to insert the additional wait state.

 The sequence of signals for this second bank 2 read is the same as the first, with the addition of one extra wait state. ALE0 is driven high one CLK cycle after the last T_c state. ALE1 remains high throughout the entire cycle.

The timing analysis for this interleaved memory system is straightforward, merely incorporating the appropriate delays through the latches and buffers.

The maximum address access time for the configuration shown in Figure 4-3 is:

	8 MHz	6 MHz
5 CLK cycles (overlapped address time)	312.5 ns	416.5 ns
− 80286 address delay (max)	− 60.0 ns	− 80.0 ns
− 8283 input-to-output delay (max)	− 22.0 ns	− 22.0 ns
− 8287 delay (max)	− 22.0 ns	− 22.0 ns
− 80286 required data setup (min)	− 10.0 ns	− 20.0 ns
Maximum address access time =	198.5 ns	272.5 ns

Maximum chip-enable access time must incorporate the additional 8205 decoder delay:

	8 MHz	6 MHz
5 CLK cycles (overlapped address time)	312.5 ns	416.5 ns
− 80286 address delay (max)	− 60.0 ns	− 80.0 ns
− 8283 input-to-output delay (max)	− 22.0 ns	− 22.0 ns
− 8205 decoder delay (max)	− 18.0 ns	− 18.0 ns
− 8287 delay (max)	− 22.0 ns	− 22.0 ns
− 80286 required data setup (min)	− 10.0 ns	− 20.0 ns
Maximum chip-enable access time =	180.5 ns	254.5 ns

If a subsequent memory access to the same memory bank occurs, an additional wait state will be inserted into the memory cycle. The maximum address access time with this interleave "hit" is:

5 CLK cycles at 8 MHz (cycle w/ 1 wait)	312.5 ns
− ALE valid delay (max)	− 20.0 ns
− 8283 STB-to-output delay (max)	− 40.0 ns
− 8287 delay (max)	− 22.0 ns
− 80286 required data setup (min)	− 10.0 ns
Maximum address access (w/ "hit") =	220.5 ns max.

Chip-enable access time with an interleave "hit" is the same as the address access time, less the 18 ns 8205 decoder delay:

Maximum chip-enable access (w/ "hit") = 202.5 ns max.

Since memory read and write commands are passed directly to the memory devices as in the standard memory interface, the command access times and command pulse widths calculated for interleaved memory systems will be the same as for non-interleaved memory systems.

If wait states are inserted into the memory cycle, access times for the relevant parameters are increased by 125 ns for each wait state (166.7 ns for waits on a 6-MHz 80286). Address, chip-enable, and command access times for a single-buffered 8-MHz system running with from zero to two wait states are shown in Table 4-6.

Interleave "hits," or back-to-back accesses to the same memory bank, typically occur during only about 7% of all bus cycles. Since an interleave hit results in an additional wait state inserted into the bus cycle, this wait state may impact overall execution time and so affect system performance. Generally, instruction prefetches by the 80286 bus unit will not result in any interleave "hits" since the prefetcher accesses sequential addresses. On average, only interleave hits that occur due to data read operations

Table 4-6. Timing for 8-MHz Interleaved Memory Operations

Maximum Access Times Before Data Valid	Number of Wait States (Assuming no "Hits")		
	Zero Wait States	One Wait State	Two Wait States
Address Access time (max)	198.5 ns	323.5 ns	448.5 ns
Chip-Select Access time (max)	180.5 ns	305.5 ns	430.5 ns
Command Access time (max)	73.0 ns	198.0 ns	323.0 ns

will directly affect 80286 execution time, as described previously; delays in data write operations do not affect the 80286 Execution unit.

Table 4-7 shows the results of several benchmarks run on an iAPX 286 system using an interleaved memory system and one using a memory system having the same number of wait states without interleaving. The values shown in the table are the ratios of execution times with interleaving to those without interleaving. In general, execution times with interleaved memory are typically only 4% more than execution times with memories having a comparable number of wait states.

MEMORY INTERFACE EXAMPLES

The following sections describe specific examples using common types of memory devices with the iAPX 286. These interface examples include read-only memories, static and pseudo-static RAM devices, and dynamic RAM devices using the 8207 Advanced Dynamic Ram Controller.

Read-Only Memory

The easiest memory devices to interface to any system are read-only memory devices (ROMs, PROMs, and EPROMs). Their byte-wide format provides a simple bus interface, and since they are read-only devices, A0 and $\overline{BHE}$ need not be included in their chip-select/enable decoding (chip-enable is similar to chip-select, but chip-enable typically determines if the device is in the active or standby power mode as well).

For a standard memory model, the address lines connected to the devices start with A1 and continue up to the maximum number the device can accept, leaving the remaining address lines for chip-enable/select decoding. To connect the devices directly to the local bus, the read-only memories must have output-enables. The output-enable is also required to avoid bus contention in some memory configurations.

Figure 4-10 shows the bus connections for read-only memories. Read-only memories require latched chip-selects. Each valid decode selects one device on the upper and lower halves of the bus to allow

Table 4-7. iAPX 286 Performance with Interleaved Memories

Program Benchmark	Ratio of Execution Time for interleaved to non-interleaved memories when most bus operations occur with:	
	0 Wait States	1 Wait State
DSO Bubble Sort (Pascal)	1.061	1.070
DSO Matrix Mult. (Pascal)	1.003	1.068
Berkeley Puzzle (Pascal)	1.020	1.028
Berkeley Sieve (C)	1.058	1.044
Quicksort (32-bit C)	1.057	1.061
Intel XFORM (ASM86)	1.007	1.007
Intel Bubble Sort (ASM86)	1.032	1.034
Average of 17 Programs	1.036	1.046

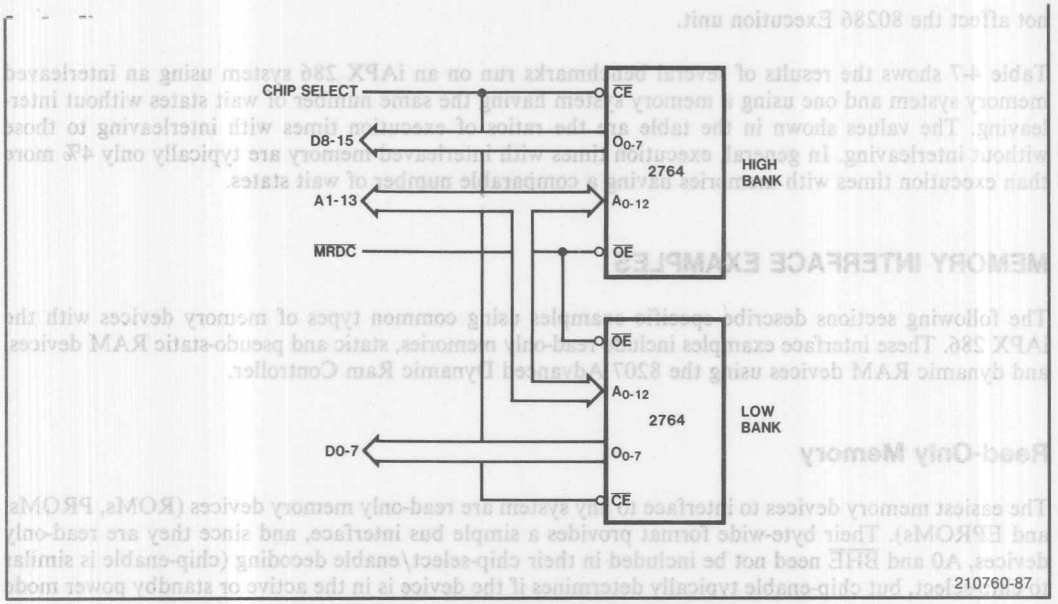


Figure 4-10. ROM/PROM/EPROM Bus Interface

byte and word access. Byte access is achieved by reading the full word onto the bus; the 80286 accepts only the appropriate byte. The output-enable is controlled by the memory read command from the Bus Controller.

TIMING ANALYSIS FOR ROMS

Read-only memories have four critical timing parameters that must be considered when interfacing these devices to an iAPX 286. These parameters are:

1. Address to Output Delay (Address Access Time)
2. Chip-Enable to Output Delay (Chip-Enable Access Time)
3. Output Enable to Output Delay (Command Access Time)
4. Output Enable High to Output Float (Data Float Time)

By comparing these memory timing requirements to the iAPX 286 timing values, the required number of wait states can be determined to guarantee a reliable memory interface.

As an example, consider a standard memory interface using 2764-20 EPROMs buffered by a single level of 8286 transceivers. Table 4-8 shows the requirements of the 2764-20 and the times allowed by an 80286 running with 1 wait state, using the standard ALE memory configuration described previously. By comparing the timing values shown in the table, it is clear that a 2764-20 EPROM is compatible with the iAPX 286 system operating with one wait state.

Timing for an interleaved memory configuration or for systems using special Strobe logic can be determined in a similar manner. It should also be noted that the timing of memory operations using any of the three memory interface techniques can be further modified by slowing the processor clock. An

Table 4-8. Timing Analysis for the 2764-20 EPROM

Timing Parameter	2764-20 EPROM (Max. Required)	8-MHz iAPX 286 1 Wait State (Min. Provided)
Address Access Time	200 ns	212.5 ns
Chip-Enable Access	200 ns	212.5 ns
Command Access Time	75 ns	185.0 ns
Data Float Time	60 ns	120.0 ns*

*Assumes circuit to delay transceiver enable if write operation to the same bus occurs after read operation.

8-MHz 80286 operating with a 15-MHz clock rather than the usual 16-MHz clock can support 200 ns memories such as the 2764-20 EPROM at zero wait states.

Table 4-9 shows the wait-state requirements for a variety of different Intel EPROMs for several different memory configurations. In each configuration, a single level of data transceivers is assumed.

Static RAM Devices

Interfacing static RAM to an 80286 CPU introduces several new requirements into the memory design. The address lines A0 and $\overline{BHE}$ must be included in the chip-select/enable decoding for the devices, and write timing must be considered in the timing analysis.

Data bus connections for each device must be restricted to either the upper- or lower-half of the data bus to allow byte transfers to only one half of the data bus. Devices like the 2114 or 2142 must not be configured to straddle both the upper and lower halves of the data bus.

To allow selection of either the upper byte, lower byte, or full 16-bit word for write operation, $\overline{BHE}$ must condition selection of the upper byte and A0 must condition selection of the lower byte. Figure 4-11 shows several techniques for selecting devices with single chip-selects and no output-enables (2114, 2141, 2147A, 2148H/49H). Figure 4-12 shows selection techniques for devices with both chip-selects and output-enables (2128, 2167A).

Devices without output-enables require inclusion of A0 and $\overline{BHE}$ to decode or enable chip selects. The $\overline{MRDC}$ and $\overline{MWRC}$ commands are used to qualify chip-select generation to prevent bus contention. Devices with common input/output pins (2114, 2142) and no output-enable require special care to strobe chip-enable low only after write-enable is valid. Write-enable and write data must be held valid until after chip-enable goes high.

For devices with separate input and output pins (2141, 2147, 2167A), the outputs can be tied together and used as described in Chapter Three.

For devices with output-enables (Figure 4-12), the write command may be gated with $\overline{BHE}$ and A0 to provide upper- and lower-bank write strobes. This simplifies chip-select decoding by eliminating $\overline{BHE}$ and A0 as a condition of decode. Although both devices are enabled during a byte write, only the appropriate high or low byte device will receive a write strobe. No bus contention exists during reads because the read command must be issued to enable the memory output drivers.

Table 4-9. Wait-State Requirements for Intel EPROMs

EPROM Device	Required Number of Wait States for 8-MHz iAPX 286		
	Standard ALE Strobe	Special Strobe Logic	Interleaved Memory System*
2732A-20	1 Wait	1 Wait	1 Wait
2732A-25	2	1	1
2732A-30	2	2	1
2732A-35	3	2	2
2732A-4	3	3	3
2764-20	1	1	1
2764-25	2	1	1
2764-30	2	2	1
2764-4	3	3	3
27128-20	1	1	1
27128-25	2	1	1
27128-30	2	2	1
27128-45	3	3	3
EPROM Device	**Required Number of Wait States for 6-MHz iAPX 286**		
	Standard ALE Strobe	**Special Strobe Logic**	**Interleaved Memory System***
2732A-20	1 Wait	0 Wait	0 Wait
2732A-25	1	1	0
2732A-30	1	1	1
2732A-35	2	1	1
2732A-4	2	2	2
2764-20	1	0	0
2764-25	1	1	0
2764-30	1	1	1
2764-4	2	2	2
27128-20	1	0	0
27128-25	1	1	0
27128-30	1	1	1
27128-45	2	2	2

*An additional wait state is required for back-to-back bus cycles to the same memory bank.

If multiple chip selects are available at the device, $\overline{BHE}$ and A0 may directly control device selection. This allows normal chip-select decoding of the address space and direct connection of the memory read and write commands to the memory devices. Another alternative is to use the multiple chip select inputs of the device to directly decode the address space (linear selects) and use the separate write-strobe technique to minimize the control circuitry needed to generate chip selects. As with the EPROMs and ROMs, the address lines connected to the memory devices must start with A1 rather than A0 for a standard memory interface.

For an interleaved memory subsystem, the low-order address lines beginning with A1 are used as bank selects. Address lines that connect to the memory devices start with the lowest address bit after the bank select address lines.

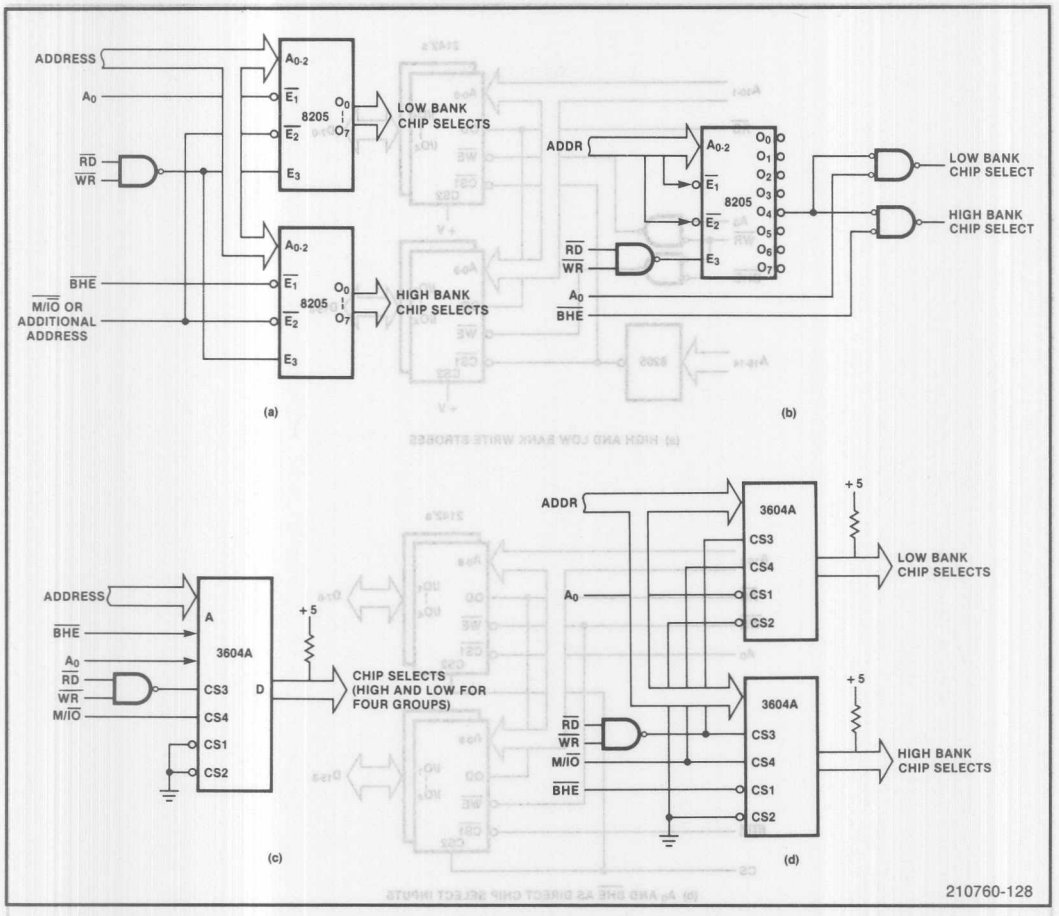


Figure 4-11. Generating Chip Selects for Devices without Output Enables

TIMING ANALYSIS FOR STATIC RAMS

When interfacing static RAM devices to an iAPX 286, write timing parameters must be considered in addition to read parameters. These additional timing parameters are:

- Address Valid to End of Write (Address Access Time)
- Chip Select to End of Write (Chip-Enable Access Time)
- Write Pulse Width (Command Access Time)
- Write Release (Address Hold From End of Write)
- Data Valid to End of Write (Write Data Access Time)
- Data Hold From End of Write (Write Data Hold Time)

By comparing these memory timing requirements to the iAPX 286 timing values, the required number of wait states can be determined to guarantee a reliable memory interface.

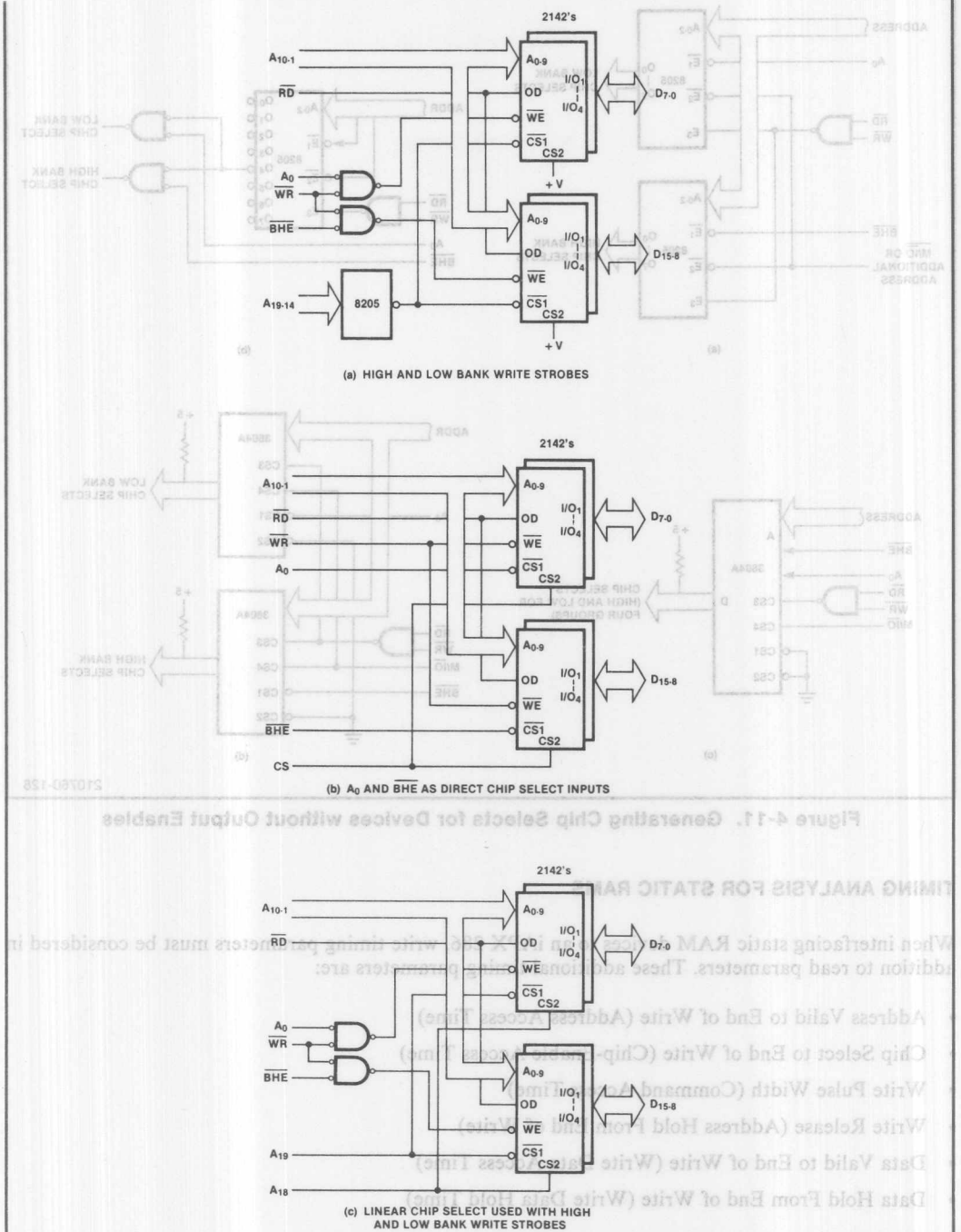


Figure 4-12. Generating Chip Selects for Devices with Output Enables

As an example, consider a standard memory interface using 2147H-1 RAM's buffered by a single level of 8286 transceivers on both inputs and outputs. Figure 4-13 shows the memory configuration used in the analysis. Table 4-10 shows the timing requirements of the 2147H-1 RAM and the times allowed by the 80286 running with zero wait states.

The Read command access time (Data valid from read) is not applicable in this configuration since WE is high throughout the cycle. $\overline{\text{MRDC}}$ merely enables the output data buffers to transfer data to the CPU. As demonstrated by the times shown in the table, the 2147H-1 static RAM is compatible with a zero-wait-state 80286 CPU.

For slower RAM devices, special address Strobe logic or an interleaved memory configuration can be used to increase the address and chip-select access times to permit operation at zero wait states.

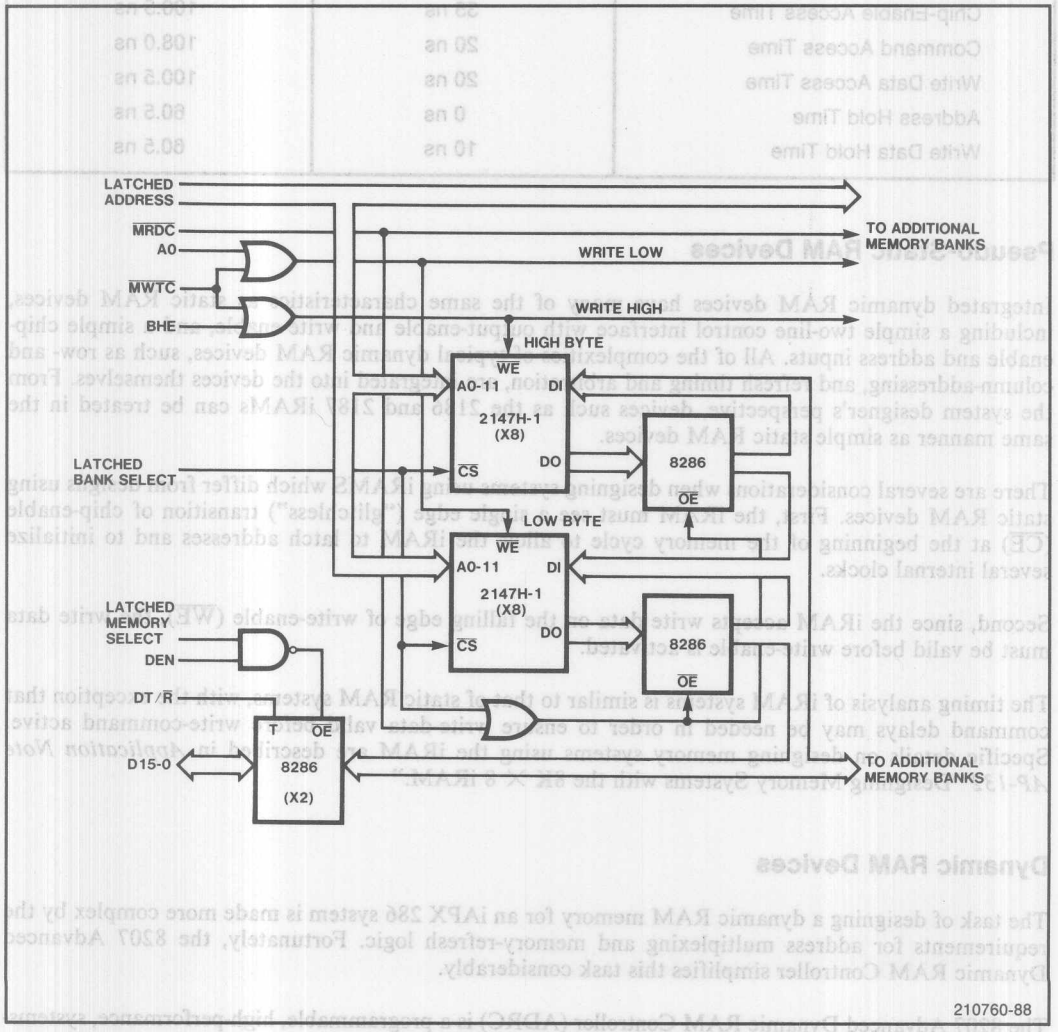


Figure 4-13. Static RAM Interface

Table 4-10. Timing Analysis for the 2147H-1 RAM

Timing	2147H-1 RAM (Max. Required)	8-MHz iAPX 286 0 Wait States (Min. Provided)
Read		
Address Access Time	35 ns	57.5 ns
Chip-Enable Access Time	35 ns	57.5 ns
Read Command Access Time	N/A	N/A
Write		
Address Access Time	35 ns	100.5 ns
Chip-Enable Access Time	35 ns	100.5 ns
Command Access Time	20 ns	108.0 ns
Write Data Access Time	20 ns	100.5 ns
Address Hold Time	0 ns	60.5 ns
Write Data Hold Time	10 ns	60.5 ns

Pseudo-Static RAM Devices

Integrated dynamic RAM devices have many of the same characteristics as static RAM devices, including a simple two-line control interface with output-enable and write-enable, and a simple chip-enable and address inputs. All of the complexities of typical dynamic RAM devices, such as row- and column-addressing, and refresh timing and arbitration, are integrated into the devices themselves. From the system designer's perspective, devices such as the 2186 and 2187 iRAMs can be treated in the same manner as simple static RAM devices.

There are several considerations when designing systems using iRAMS which differ from designs using static RAM devices. First, the iRAM must see a single edge ("glitchless") transition of chip-enable ($\overline{CE}$) at the beginning of the memory cycle to allow the iRAM to latch addresses and to initialize several internal clocks.

Second, since the iRAM accepts write data on the falling edge of write-enable ($\overline{WE}$), the write data must be valid before write-enable is activated.

The timing analysis of iRAM systems is similar to that of static RAM systems, with the exception that command delays may be needed in order to ensure write-data valid before write-command active. Specific details on designing memory systems using the iRAM are described in *Application Note AP-132* "Designing Memory Systems with the 8K × 8 iRAM."

Dynamic RAM Devices

The task of designing a dynamic RAM memory for an iAPX 286 system is made more complex by the requirements for address multiplexing and memory-refresh logic. Fortunately, the 8207 Advanced Dynamic RAM Controller simplifies this task considerably.

The 8207 Advanced Dynamic RAM Controller (ADRC) is a programmable, high-performance, systems-oriented controller designed to easily interface 16K, 64K, and 256K dynamic RAMs to a wide range

of microprocessors, including the 80286 CPU. Dual-port configurations are supported by two independent ports that operate in both synchronous- and asynchronous-processor environments. The 8207 also supports error detection and correction using the 8206 Error Detection and Correction Unit (EDCU).

The 8207 Controller timing is optimized for maximum performance. With fast dynamic RAM (such as 2118-10's) the 8207 can run at 0 wait states for most bus operations.

As shown in Figure 4-14, the 8207 can be controlled directly by the 80286 status lines or by 82288 command outputs, and can operate synchronously or asynchronously to the system clock. Figure 4-15 provides a block diagram for a single-port 8207 memory subsystem using the command interface. PCTL is tied low to configure the 8207 in command mode for compatibility with the 80286 signals. CPU status signals directly drive the read and write inputs to the controller while the 82284 CLK signal provides the synchronous clock. Address lines connect directly to the controller, while data transfer is controlled by two sets of latches between the memory banks and the CPU. A byte mark latch decodes A0 and $\overline{BHE}$ to condition the write enable signals for the high- and low-byte memory banks.

The 8207 is capable of addressing 16K, 64K, and 256K dynamic RAMs, using an interleaved memory arrangement. Memory can be divided into four banks, with each bank having its own $\overline{RAS}$ and $\overline{CAS}$ signal pair. Low-order address lines serve as bank selects to allow interleaved memory cycles. (Figure 4-16 shows the address connections to the ADRC for all three types of RAM devices.)

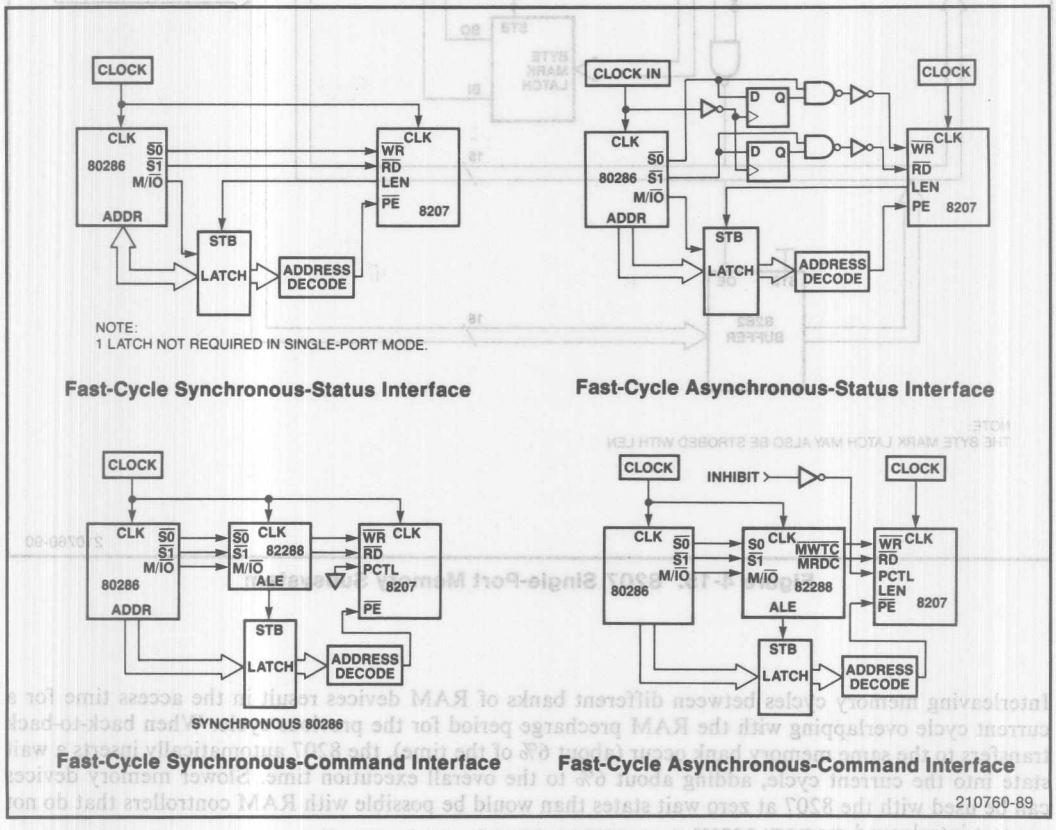


Figure 4-14. 80286-8207 Interfaces

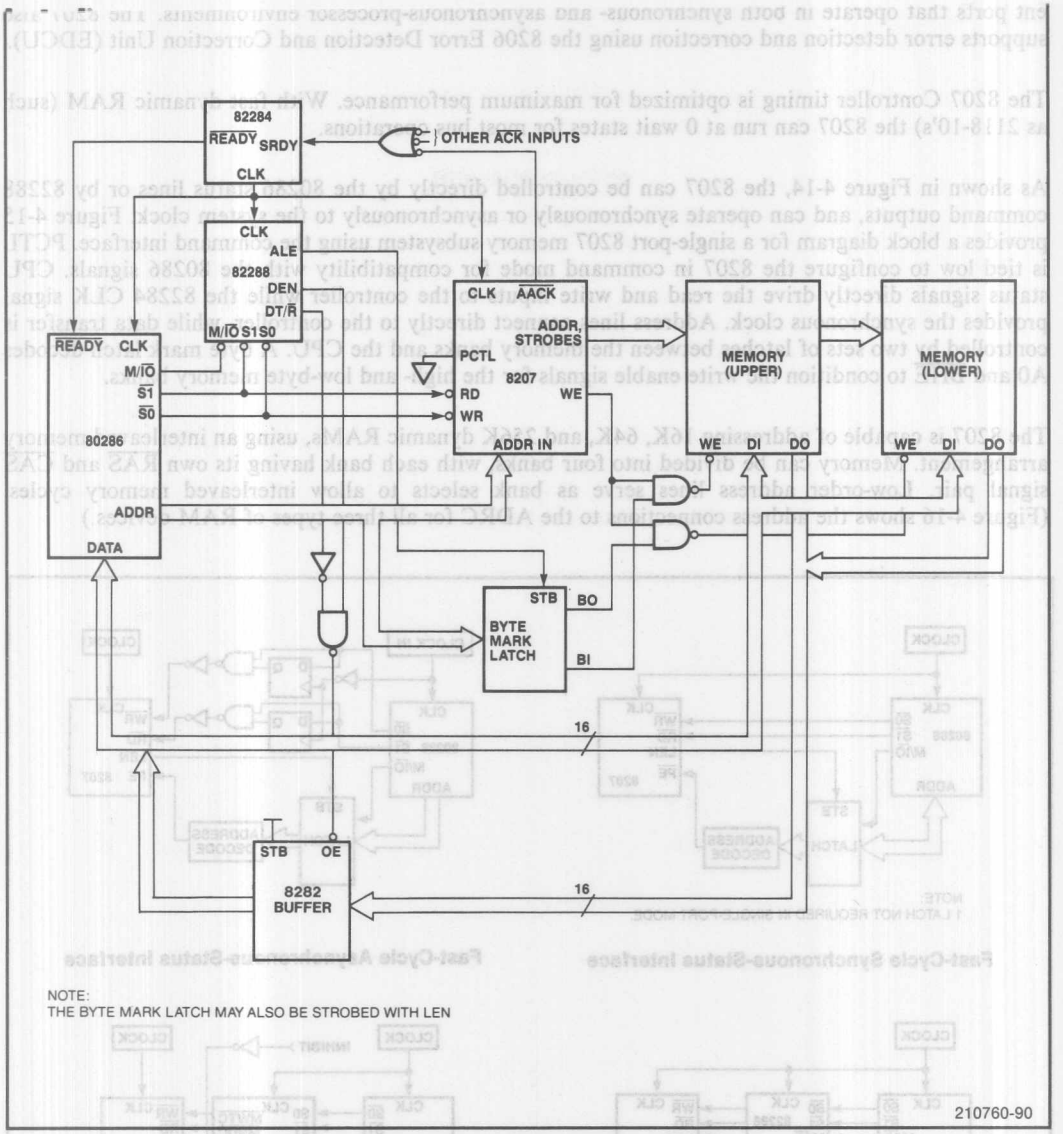


Figure 4-15. 8207 Single-Port Memory Subsystem

Interleaving memory cycles between different banks of RAM devices result in the access time for a current cycle overlapping with the RAM precharge period for the previous cycle. When back-to-back transfers to the same memory bank occur (about 6% of the time), the 8207 automatically inserts a wait state into the current cycle, adding about 6% to the overall execution time. Slower memory devices can be used with the 8207 at zero wait states than would be possible with RAM controllers that do not support interleaved memory access.

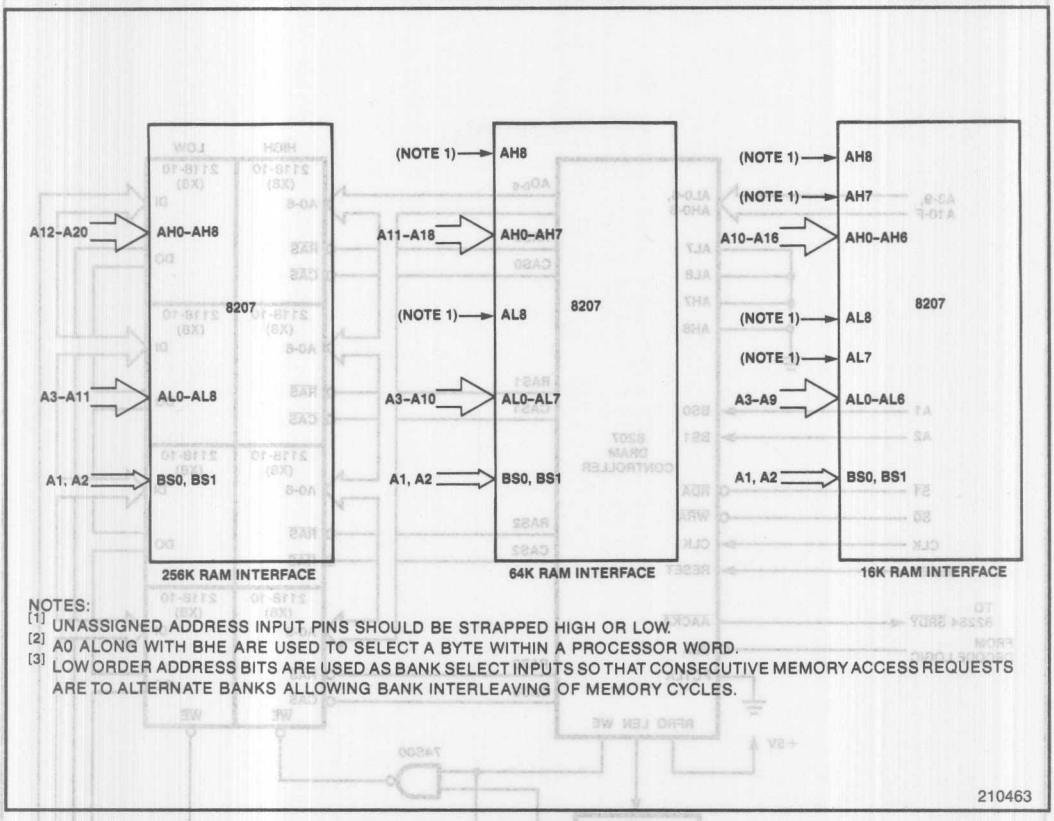


Figure 4-16. 256K, 64K, and 16K RAM Address Connections

TIMING ANALYSIS FOR THE 8207 RAM CONTROLLER

The analysis of timing for dynamic RAMs is more extensive than for a static RAM or EPROM. The analysis must include RAS, CAS, and refresh timing as well as standard bus interface timing. Figure 4-17 shows the memory subsystem that serves as the model for the timing analysis in this section.

The subsystem shown in Figure 4-17 is a single-port configuration designed with an 8207 ADRC and four banks of 2118-10 16K x 1 bit dynamic RAMs. Each bank is sixteen bits wide and is divided into a high byte (D15-8) and a low byte (D7-0). Most 8207 signals connect directly to 80286, 82288, and 82284 inputs and outputs. No address buffering is required. The RAM data inputs and outputs are buffered, with the 82288 DEN and DT/$\overline{R}$ signals controlling the output data buffers. The $\overline{PEA}$ (Port Enable A) input to the ADRC is a non-latched chip-select driven by address-decode logic.

The 8207 is programmed at reset through strapping options and a 16-bit program word. $\overline{PCTLA}$, $\overline{PCTLB}$, and RFRQ are strapped high or low to program port and refresh options. $\overline{PCTLA}$ is tied low to select command mode. ($\overline{PCTLB}$ is not shown; $\overline{PEB}$ should be strapped high to ignore the port B interface.) RFRQ is tied high to program the 8207 for self-refresh mode. PDI (Program Data Input) accepts a 16-bit serial bit stream containing the program word. By strapping PDI high or low, one of two sets of default parameters are selected. The example circuit in Figure 4-17 is programmed with the non-ECC mode default parameters shown in Table 4-11.

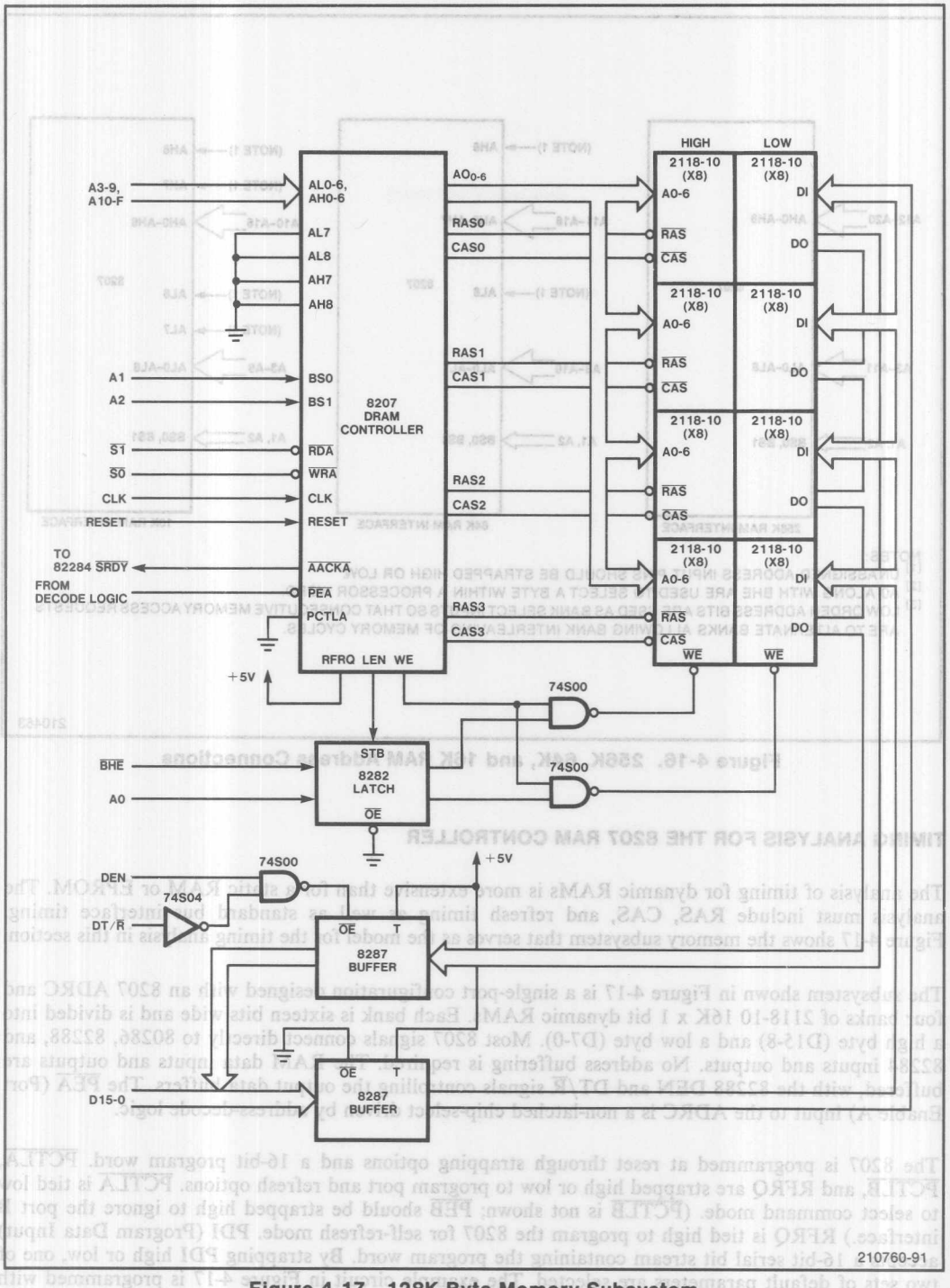


Figure 4-17. 128K-Byte Memory Subsystem

Table 4-11. Non-ECC Mode Programming, PDI Pin (57) Tied to Ground

Port A is Synchronous
Port B is Asynchronous
Fast-cycle Processor Interface (80286)
Fast RAM (2118-10 compatible)
Refresh Interval uses 236 clocks
128 Row refresh in 2 ms; 256 Row refresh in 4 ms.
Fast Processor Clock Frequency (16 MHz)
"Most Recently Used" Priority Scheme
4 RAM Banks Occupied

As shown in Figure 4-18, PDI can be driven by a 16-bit serial shift register to program the 8207 for non-default operation. Sixteen bits are loaded into the register in parallel during reset and are shifted out in serial by the 8207 program clock (PCLK).

Figure 4-19 shows the timing for three back-to-back memory cycles: a read from bank 1, a write to bank 2, and a read from bank 2. Since the first two cycles are to different memory banks, the $\overline{RAS}$ for the second cycle is overlapped with the $\overline{RAS}$ precharge period for the first cycle to allow the subsystem to respond with no wait states.

In the case of back-to-back cycles to the same bank (cycles 2 and 3), the controller automatically waits for the $\overline{RAS}$ precharge period between cycles before asserting $\overline{RAS}$. $\overline{AACKA}$ from the 8207, which must be connected to the 82284 $\overline{SRDY}$ input, is delayed long enough to guarantee an additional wait state to compensate for the slower subsystem response time. The 82288 $\overline{SRDY}$ input is used rather than the ARDY input to accommodate the critical $\overline{AACKA}$ signal timing.

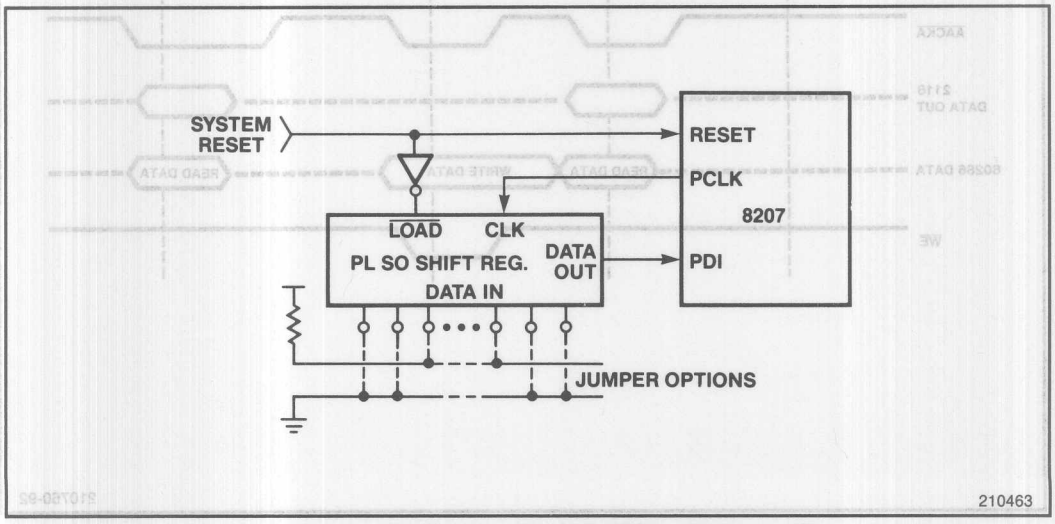


Figure 4-18. External Shift Register Interface

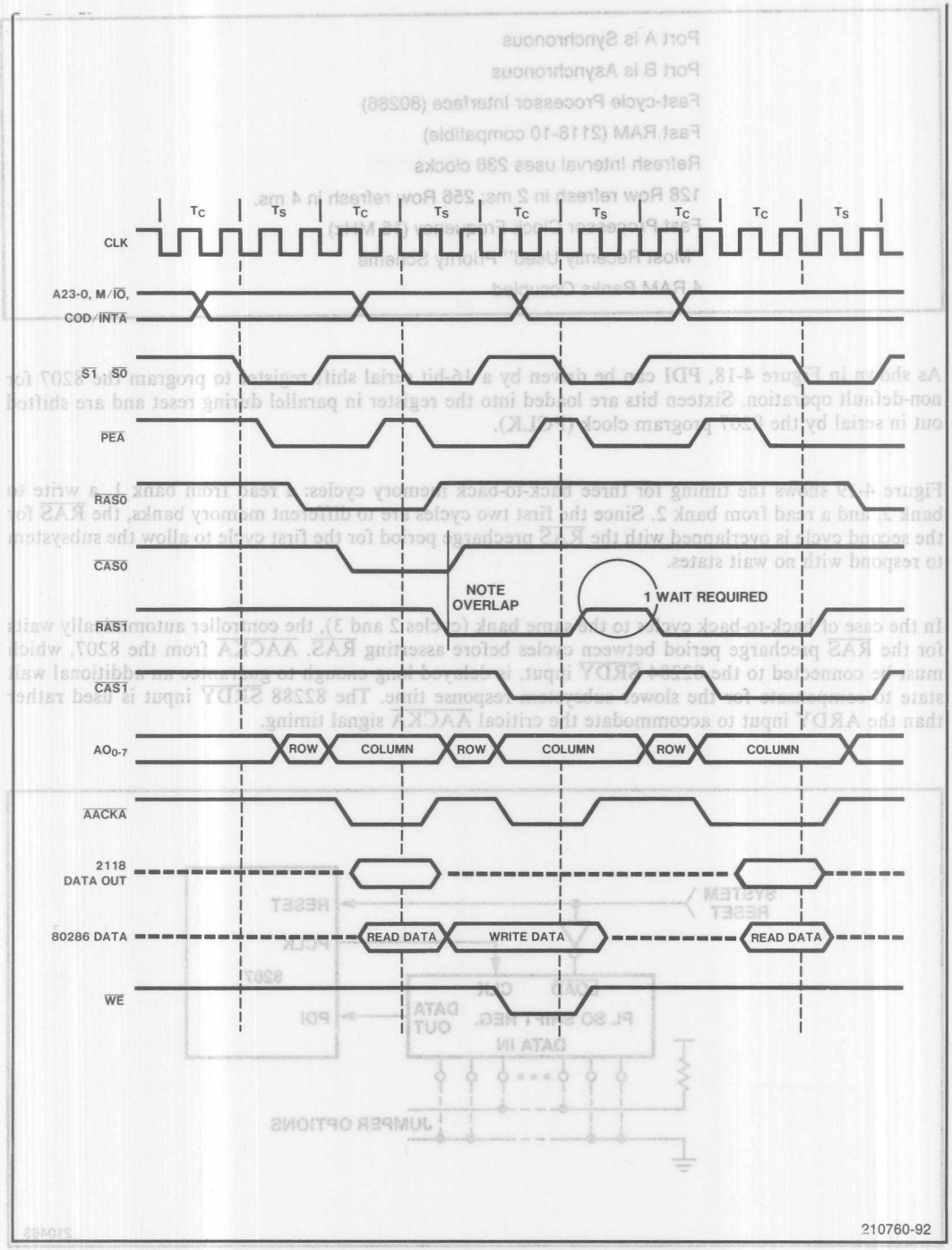


Figure 4-19. 128K-Byte Memory Subsystem Timing

RAM timing to determine device compatibility is measured from $\overline{RAS}$ going low to data valid at the CPU pins. Maximum allowed $\overline{RAS}$ access time is:

	8 MHz	6 MHz
3 CLK cycles	187.5 ns	250 ns
− $\overline{RAS}$ active delay (max)	− 35.0 ns	− 35 ns
− 8287 transceiver delay (max)	− 22.0 ns	− 22 ns
− 80286 required data setup (min)	− 10.0 ns	− 20 ns
Max. allowed $\overline{RAS}$ access time =	120.5 ns	173 ns

The corresponding $\overline{CAS}$ access time is:

	8 MHz	6 MHz
2 CLK cycles	125 ns	166.6 ns
− $\overline{CAS}$ active delay (max)	− 35 ns	− 35.0 ns
− 8287 transceiver delay (max)	− 22 ns	− 22.0 ns
− 80286 required data setup (min)	− 10 ns	− 20.0 ns
Max. allowed $\overline{CAS}$ access time =	58 ns	89.6 ns

For typical DRAM devices, the $\overline{CAS}$ access time will be the limiting parameter. Table 4-12 shows the timing requirements of the 2118-10 DRAM and the times allowed by the 80286 and 8207 operating with zero wait states.

The timing values shown in the table indicate that the 2118-10 DRAM is compatible with an 8-MHz iAPX 286 and 8207 operating at zero wait states. The circuit shown in Figure 4-19 will run at 0 wait states for most memory cycles; the subsystem will run with 1 wait state only when back-to-back accesses occur to the same memory bank.

With 2118-15 or 2164A-15 devices, this same circuit can be run at 1 wait state for most bus cycles on an 8-MHz system (select slow RAM option in program word). For 6-MHz iAPX 286 systems, 2118-15 or 2164A-15 memories can be used at zero wait states, as shown in Table 4-13. Detailed timing specifications for the 8207 DRAM Controller are provided in the Appendices.

Table 4-12. 8-MHz Timing Analysis for the 2118-10 DRAM

Timing Parameter	2118-10 DRAM (Max. Required)	8-MHz iAPX 286 with 8207 0 Wait States (Min. Provided)
$\overline{RAS}$ Access Time	100 ns	120.5 ns
$\overline{CAS}$ Access Time	55 ns	58 ns

Table 4-13. 6-MHz Timing Analysis for the 2164A-15 DRAM

Timing Parameter	2164A-15 DRAM (Max. Required)	6-MHz iAPX 286 with 8207 0 Wait States (Min. Provided)
$\overline{RAS}$ Access Time	150 ns	173 ns
$\overline{CAS}$ Access Time	85 ns	89.6 ns

ERROR CORRECTION USING THE 8206 EDCU

The 8207 RAM Controller supports error detection and correction using the 8206 Error Detection and Correction Unit (EDCU). Figure 4-20 shows a memory subsystem using the 8206 EDCU with the 8207 RAM Controller.

The 8207 interface to the RAM devices in ECC mode is almost identical to that in non-ECC mode. RAM is divided into four banks with twenty-two devices in each bank. Sixteen RAM devices store data for use by the CPU; the remaining six RAM devices store check bits for use by the 8206 (EDCU).

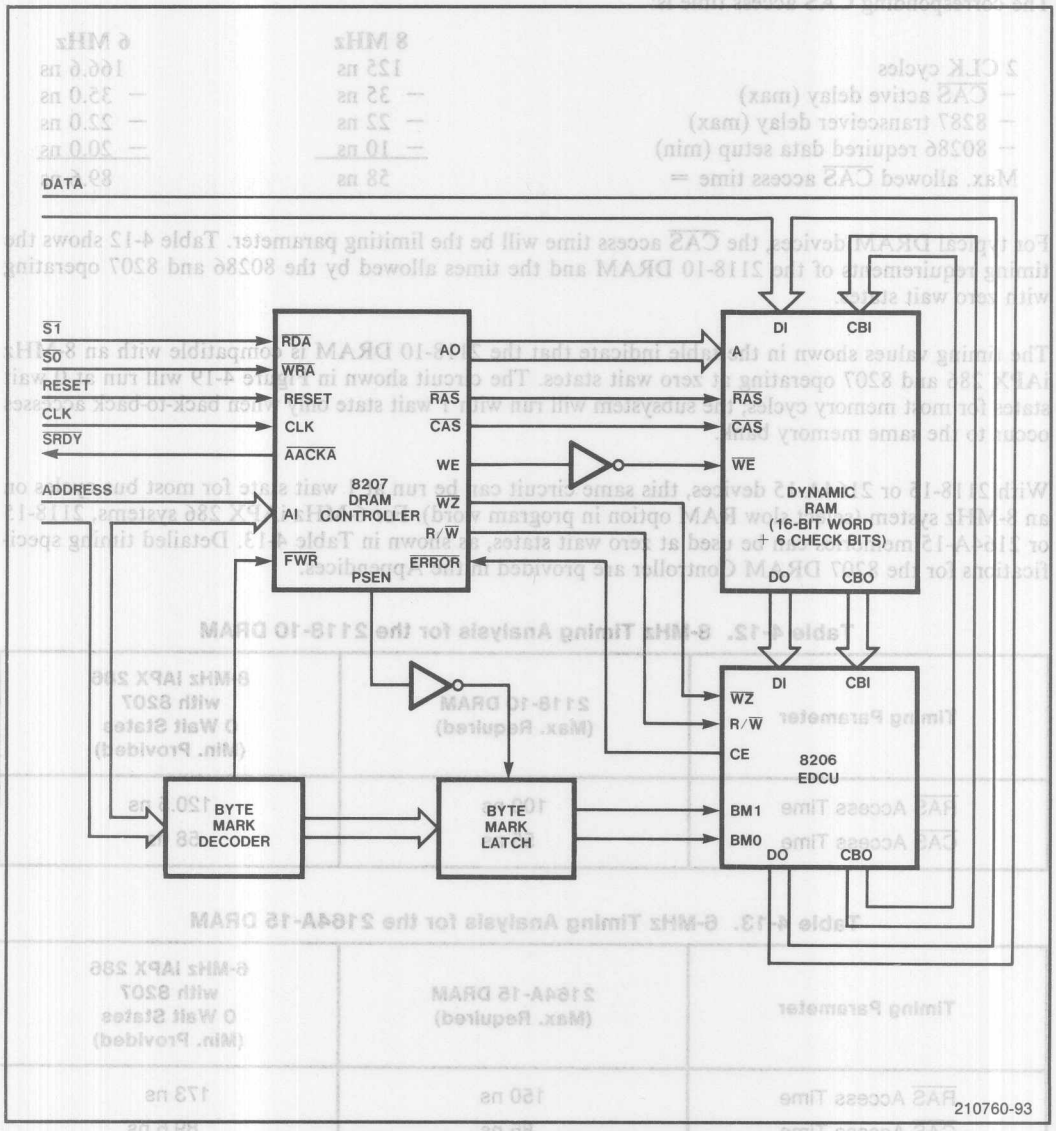


Figure 4-20. 8207 Memory Subsystem with ECC

During data writes, the EDCU computes a check code and writes the code into the check RAM. During data reads, the EDCU compares the read data to the check code to detect errors. Single-bit errors are corrected and double-bit or multiple-bit errors are flagged. The 8206 $\overline{\text{ERROR}}$ and CE (Correctable Error) signals inform the 8207 of error status. R/$\overline{\text{W}}$ (Read/$\overline{\text{Write}}$) and WZ (Write Zero) from the 8207 control write/read and initialization modes of the 8206. Byte marks determine whether a byte or word write is performed by the 8206.

It is important to note that the 8206 EDCU introduces a propagation delay of 67 ns (max) into memory cycles. The ADRC automatically compensates for the delay by inserting an additional wait state into all bus cycles to ECC-protected memory.

DUAL-PORT MEMORY SUBSYSTEMS USING THE 8207 ADRC

The 8207 RAM Controller contains on-chip arbitration logic to control dual-port memory subsystems. The concept of a dual-port memory subsystem, permitting access by an 80286 over the 80286 local bus and also access by other processors over a system bus, was introduced in Chapter Two.

In dual-port mode, the 8207 must arbitrate memory requests between port A, port B, and port C—the internal refresh port. Two port priority options can be programmed to select which port has priority over the others in gaining access to the RAM memory array. In addition, the 8207 supports a $\overline{\text{LOCK}}$ input to prevent a second port from gaining access to the memory array until the port asserting the $\overline{\text{LOCK}}$ request has released the memory.

Chapter Seven contains a detailed explanation of how to configure a dual-port subsystem between the 80286 local bus and the Multibus system bus. Specific examples are given for a dual-port memory subsystem with error checking and correction, and specific guidelines are given for implementing the $\overline{\text{LOCK}}$ input to the 8207 to prevent deadlock situations.

During data writes, the EDCU computes a check code and writes the code into the check RAM. During data reads, the EDCU compares the read data to the check code to detect errors. Single-bit errors are corrected and double-bit or multiple-bit errors are flagged. The 8206 $\overline{\text{ERROR}}$ and CE (Correctable Error) signals inform the 8207 of error status. R/$\overline{\text{W}}$ (Read/$\overline{\text{Write}}$) and WZ (Write Zero) from the 8207 control write/read and initialization modes of the 8206. Byte mark/s determine whether a byte or word write is performed by the 8206.

It is important to note that the 8206 EDCU introduces a propagation delay of 67 ns (max) into memory cycles. The ADRC automatically compensates for the delay by inserting an additional wait state into all bus cycles to ECC-protected memory.

DUAL-PORT MEMORY SUBSYSTEMS USING THE 8207 ADRC

The 8207 RAM Controller contains on-chip arbitration logic to control dual-port memory subsystems. The concept of a dual-port memory subsystem, permitting access by an 80286 over the 80286 local bus and also access by other processors over a system bus, was introduced in Chapter Two.

In dual-port mode, the 8207 must arbitrate memory requests between port A, port B, and port C—the internal refresh port. Two port priority options can be programmed to select which port has priority over the others in gaining access to the RAM memory array. In addition, the 8207 supports a $\overline{\text{LOCK}}$ input to prevent a second port from gaining access to the memory array until the port asserting the $\overline{\text{LOCK}}$ request has released the memory.

Chapter Seven contains a detailed explanation of how to configure a dual-port subsystem between the 80286 local bus and the Multibus system bus. Specific examples are given for a dual-port memory subsystem with error checking and correction, and specific guidelines are given for implementing the $\overline{\text{LOCK}}$ input to the 8207 to prevent deadlock situations.

I/O Interfacing

5

I/O INTERFACING

The iAPX 286 supports both 8-bit and 16-bit input/output devices that can be mapped into either a special I/O-address space or the iAPX 286 memory-address space.

- I/O-mapping permits I/O devices to reside in a separate 64K I/O address space. Four special I/O instructions are available for device communications, and the I/O Privilege level offers I/O device protection.
- Memory-mapping permits I/O devices to reside anywhere in the iAPX 286 memory space. The full 80286 instruction set can be used for I/O operations, and the full memory-management and protection features of the iAPX 286 can be used for device protection.

This chapter describes how to design I/O devices into an iAPX 286 system:

- The first section of this chapter establishes the tradeoffs between I/O-mapped and memory-mapped I/O.
- The second section describes several techniques for connecting data buses and generating chip-selects for both 8-bit and 16-bit I/O devices. Various alternatives are described for both I/O-mapped and memory-mapped devices.
- The third section discusses the timing considerations for I/O operations, and contains examples of worst-case timing analyses.
- The fourth section contains specific I/O interface examples, including interfaces for the 8274 Multi-Protocol Serial Controller, the 8255A Programmable Peripheral Interface, and the 8259A Programmable Interrupt Controller.
- The last section of this chapter introduces the iSBX bus interface; the iSBX bus allows single-board computer systems to be modularly expanded simply by plugging in I/O-mapped iSBX Multimodule Boards.

I/O-MAPPING VS. MEMORY-MAPPING

The 80286 CPU is capable of supporting either I/O-mapped or memory-mapped I/O devices. As outlined in the introduction, these two alternatives differ in three key respects. Although a decision to use one alternative or another will depend on the particular requirements of the design, the tradeoffs between these alternatives should be considered.

Address-Decoding

I/O-mapped devices reside in a separate 64K I/O address space, whereas memory-mapped devices reside in the full 16 Mbyte physical memory space. For this reason, the address-decoding required to generate chip-selects for I/O-mapped devices may be simpler than that required for memory-mapped devices.

iAPX 286 Instruction Set

I/O-mapped devices are accessible to programmers using the IN, OUT, INS, and OUTS instructions of the iAPX 286. IN and OUT instructions transfer data between I/O addresses and the AX (16-bit

transfers) or AL (8-bit transfers) registers. The first 256 bytes of I/O space are directly-addressable by the I/O instructions, whereas the entire I/O space (64K bytes) can be indirectly-addressed through the DX register. INS and OUTS are string instructions used to transfer blocks of data between memory and I/O devices. For more information on these I/O instructions, see the iAPX 286 Programmer's Reference Manual.

Memory-mapped devices are accessible using the full iAPX 286 instruction set, allowing efficient coding of such tasks as I/O-to-memory, memory-to-I/O, and I/O-to-I/O transfers, as well as compare and test operations.

Figure 5-1 illustrates the advantages of using memory-mapped I/O over I/O-mapped I/O in performing a simple bit-manipulation task. After setting up pointers to the I/O device, the memory-mapped I/O example accomplishes the task in a single instruction; the I/O-mapped example requires three.

Device Protection

I/O-mapped devices are offered protection by the iAPX 286 I/O Privilege level. This privilege level can be used either to prevent a task from accessing any I/O devices, or to permit the task access to all of the I/O-mapped devices.

Memory-mapped devices fall under the protection of the iAPX 286 memory-management and protection features. Depending on how devices are mapped into the memory space, individual tasks may be given access to particular I/O devices, while other devices are either visible-but-protected, or else mapped entirely out of the task's visible address space. Memory-mapping of devices thus permits more flexibility in offering protection of individual system resources than I/O-mapping does.

BIT-MANIPULATION FOR
I/O-MAPPED I/O DEVICE

```
SETBIT:  MOV   DX,STAT_REG  ; point to status

         IN    AL,DX        ; read status word
         OR    AL,10H       ; set bit 4
         OUT   DX,AL        ; output status word
```

BIT-MANIPULATION FOR
MEMORY-MAPPED I/O DEVICE

```
SETBIT:  MOV   AX,IO_SET_BASE ; I/O base addr
         MOV   ES,AX          ; load seg base

         OR    ES:BYTE PTR STAT_REG,10H
                 ; set bit 4 of status word
```

Figure 5-1. Comparing Memory-Mapped and I/O-Mapped I/O

INTERFACING TO 8-BIT AND 16-BIT I/O

Although the iAPX 286 can address I/O devices as either byte (8-bit) or word (16-bit) devices, several considerations must be observed when connecting these devices to the iAPX 286 bus. The following sections describe data bus connections and chip-select techniques for both I/O-mapped and memory-mapped I/O devices.

Address Decoding

As mentioned in the previous section, the address-decoding for I/O-mapped devices is somewhat simpler than that for memory-mapped devices. One simple technique for decoding memory-mapped I/O addresses is to map the entire iAPX 286 I/O space into a 64-Kilobyte region of the iAPX 286 memory space. If desired, the address decoder can be configured so that the I/O devices respond to *both* a memory address and an I/O address. This configuration allows system compatibility with software using the iAPX 86-family I/O instructions, while also permitting the use of software that takes advantage of memory-mapped I/O and the iAPX 286 memory-management and protection features.

Another factor affecting decoder complexity is the particular choice of addresses for either I/O-mapped or memory-mapped devices. Before selecting addresses for various I/O devices, however, designers must be aware of several restricted address spaces imposed by the iAPX 286 architecture. Figure 5-2 shows these restricted address regions for both memory-mapped and I/O-mapped devices.

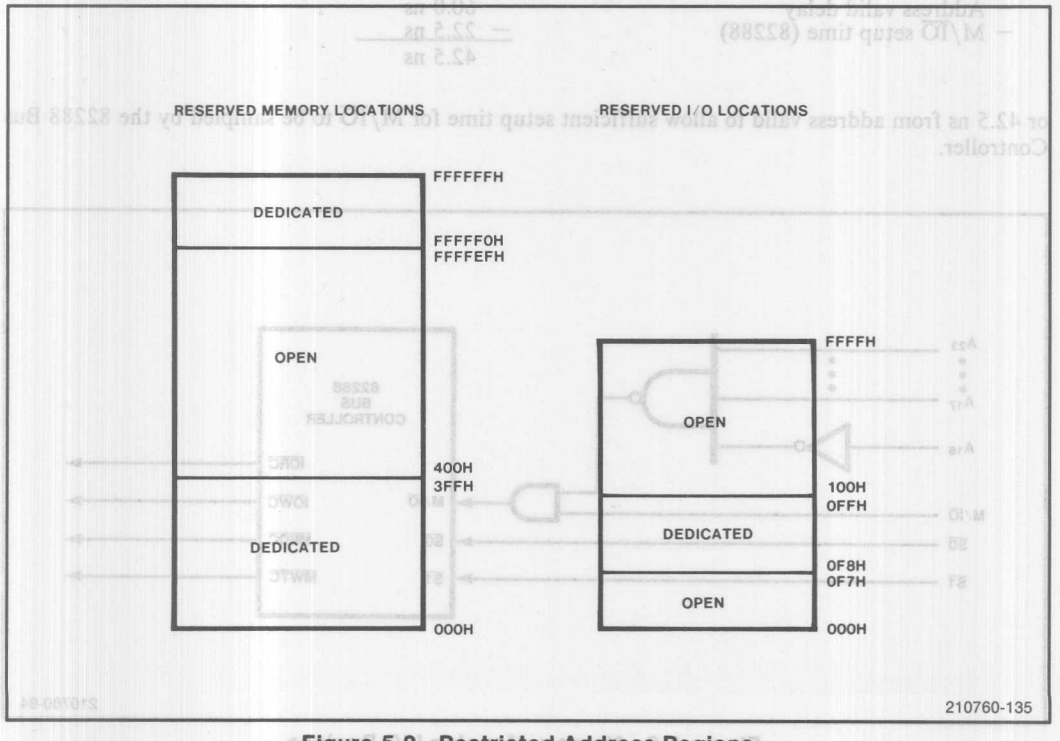


Figure 5-2. Restricted Address Regions

complex by decoding fewer of the lower address lines, resulting in larger granularity in the addresses corresponding to an individual I/O device. The 64-K I/O address space of the iAPX 286 leaves plenty of freedom for allocating addresses to individual I/O devices.

Memory-Mapped I/O

Figure 5-3 shows a simple decoding circuit for mapping the iAPX 286 I/O space into a 64K region of the iAPX 286 memory space. This technique, described above, allows individual I/O devices to respond to both a memory address and an I/O address. This particular circuit maps the I/O space into the region of the iAPX 286 memory space which corresponds to addresses FE 0000H through FE FFFFH (this region is the second 64K region from the top of the physical address space; the top 64K region typically contains the code for iAPX 286 system-reset processing).

When this memory-mapping technique is used, the low sixteen address bits must still be decoded to generate chip selects (in the same manner as for I/O-mapped devices). The $\overline{\text{IORC}}$ and $\overline{\text{IOWC}}$ commands, rather than $\overline{\text{MRDC}}$ and $\overline{\text{MWTC}}$, are generated by the Bus Controller whenever a bus operation accesses addresses in the specified region. For this reason, subsequent sections of this chapter will not distinguish between memory-mapped and I/O-mapped devices.

The maximum decode time for the simple gating arrangement shown in Figure 5-3 is:

2 CLK cycles at 8 MHz	125.0 ns
− Address valid delay	− 60.0 ns
− M/$\overline{\text{IO}}$ setup time (82288)	− 22.5 ns
	42.5 ns

or 42.5 ns from address valid to allow sufficient setup time for M/$\overline{\text{IO}}$ to be sampled by the 82288 Bus Controller.

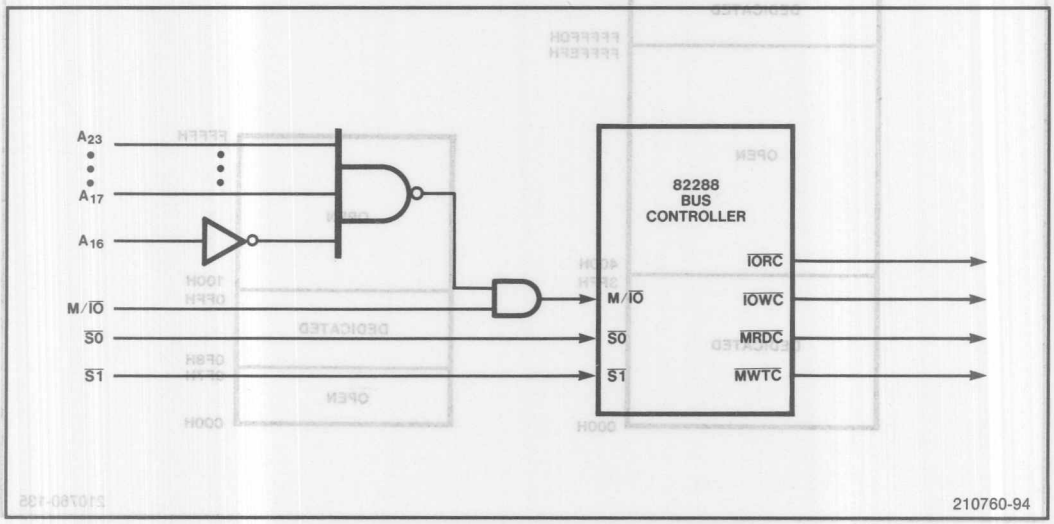


Figure 5-3. Memory-Mapping I/O Devices

The same circuit technique shown in Figure 5-3 may be used to memory-map I/O devices that reside on the Multibus. The Bus Controller shown in the circuit would then be used to control the Multibus signals; this technique is discussed further in Chapter Seven in conjunction with Multibus I/O.

8-Bit I/O

Although 8-bit I/O devices may be connected to either the upper- or lower-half of the data bus, it is recommended that designers use the lower half of the data bus for 8-bit devices.

The particular address assigned to a device determines whether byte transfers will use the upper- or lower-half of the data bus.

- If a device is connected to the upper half of the data bus, all I/O addresses assigned to the device must be odd (A0 = 1).
- If the device is on the lower half of the bus, its addresses must be even (A0 = 0).

Since A0 will always be high or low for a specific device, this address line cannot be used as an address input to select registers within a device. If a device on the upper half and a device on the lower half of the bus are assigned addresses that differ only in A0 (adjacent odd and even addresses), A0 and BHE must be conditions of chip select decode to prevent a write to one device from erroneously performing a write to the other. Figure 5-4 shows several techniques for generating chip selects for I/O-mapped devices.

The first technique (a) uses separate 8205's to generate chip selects for odd- and even-addressed byte peripheral devices. If a word transfer is performed to an even-addressed device, the adjacent odd-addressed I/O device is also selected. This allows accessing the devices individually for byte transfers or simultaneously as a 16-bit device for word transfers. The second technique (b) restricts the chip selects to byte transfers. Word transfers to odd addresses, since they are performed as two byte transfers, are also permitted. The last technique (c) uses a single 8205 to generate odd and even device selects for byte transfers. Even device selects are generated for both byte and word transfers to even addresses.

If greater than 256 bytes of I/O space are required, additional decoding beyond what is shown in the examples may be necessary. This can be done with additional TTL, 8205's, or PROMs. Figure 5-5 shows an Intel 3605A bipolar PROM used as an I/O address decoder generating latched chip-selects. The bipolar PROM is slightly slower than multiple levels of TTL (50 ns for PROM vs. 30 to 40 ns for TTL), but provides full decoding in a single package, and allows easy reconfiguration of the system I/O map by inserting a new PROM; no circuit board or wiring modifications are required. By using ALE to latch the decoded chip selects, up to 68 ns are available for decoding without affecting address access timing:

2 CLK cycles at 8 MHz	125 ns
minus address valid delay (max)	− 60 ns
plus ALE delay (min)	+ 3 ns
Minimum address decode time =	68 ns max

One last technique for interfacing with 8-bit peripherals is considered in Figure 5-6. The 16-bit data bus is multiplexed onto an 8-bit bus to accommodate byte-oriented DMA or block transfers to memory-mapped 8-bit I/O. Devices connected to this interface may be assigned a sequence of odd and even addresses rather than all odd or all even.

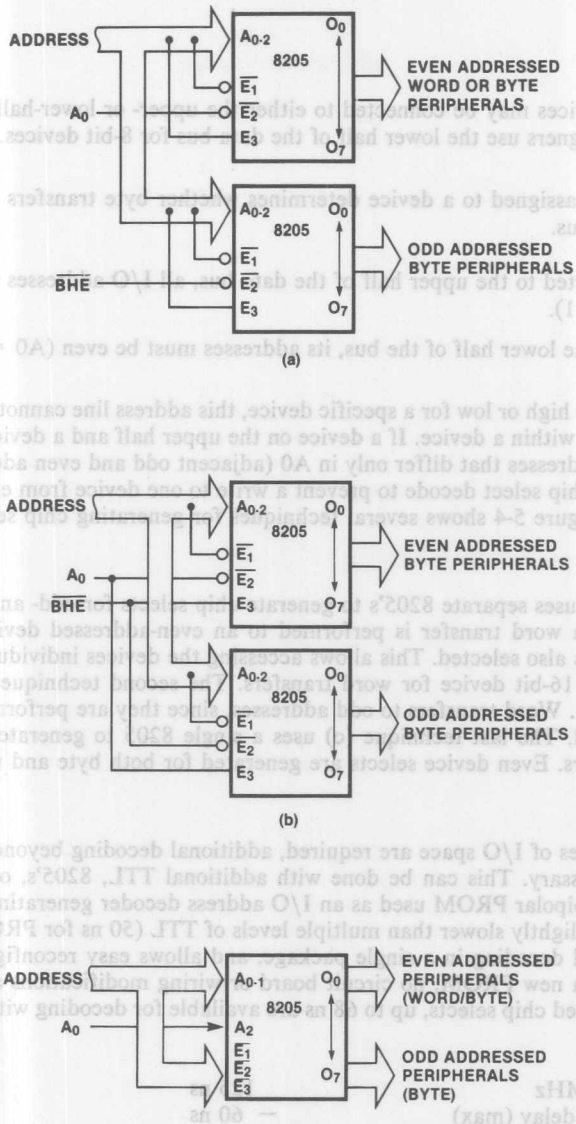


Figure 5-4. Generating I/O Chip Selects

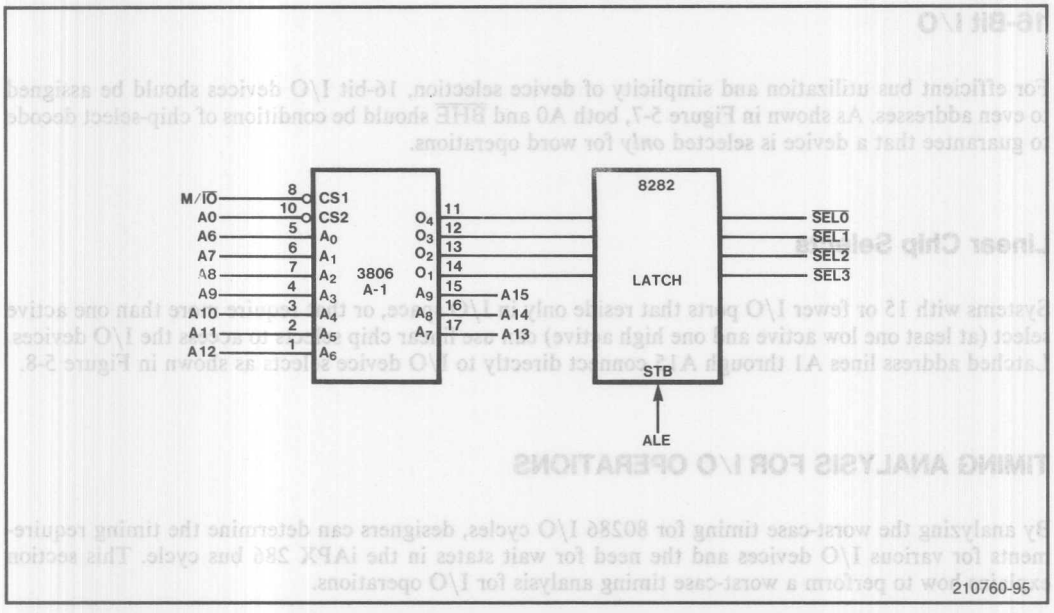


Figure 5-5. Bipolar PROM Decoder for 8-bit or 16-bit I/O Devices

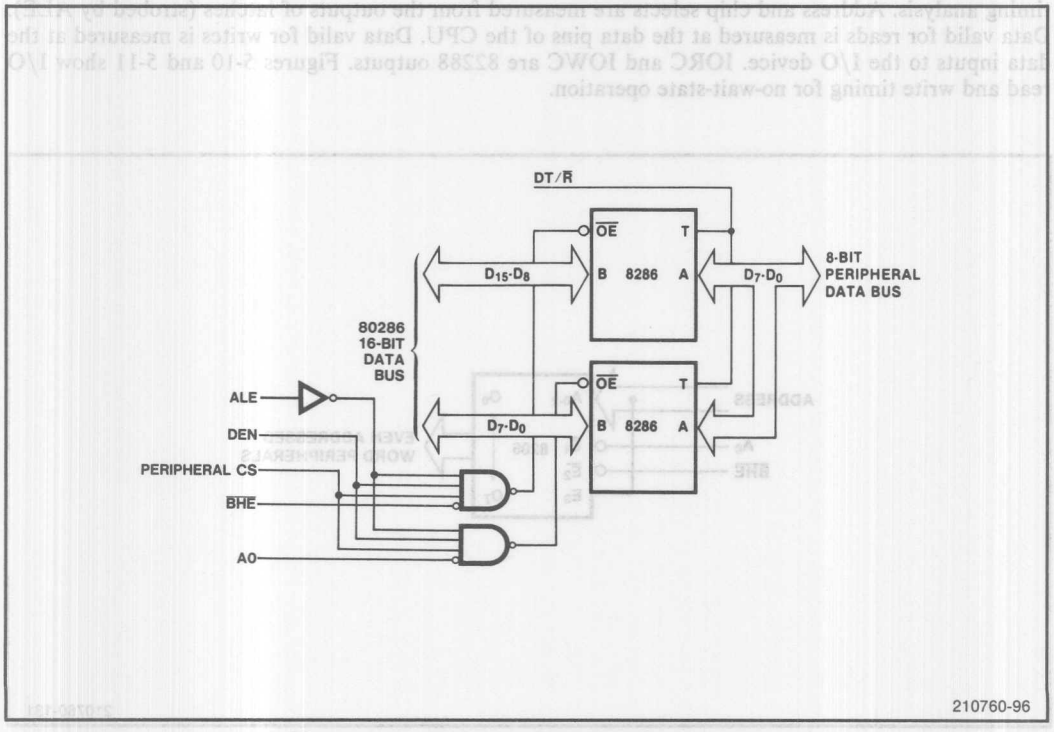


Figure 5-6. 16-bit to 8-bit Bus Conversion

For efficient bus utilization and simplicity of device selection, 16-bit I/O devices should be assigned to even addresses. As shown in Figure 5-7, both A0 and $\overline{BHE}$ should be conditions of chip-select decode to guarantee that a device is selected *only* for word operations.

Linear Chip Selects

Systems with 15 or fewer I/O ports that reside only in I/O space, or that require more than one active select (at least one low active and one high active) can use linear chip selects to access the I/O devices. Latched address lines A1 through A15 connect directly to I/O device selects as shown in Figure 5-8.

TIMING ANALYSIS FOR I/O OPERATIONS

By analyzing the worst-case timing for 80286 I/O cycles, designers can determine the timing requirements for various I/O devices and the need for wait states in the iAPX 286 bus cycle. This section explains how to perform a worst-case timing analysis for I/O operations.

Timing for 80286 I/O cycles is identical to memory cycle timing in most respects and, like memory timing, is dependent on a particular model. Figure 5-9 shows the model used here to discuss worst-case timing analysis. Address and chip selects are measured from the outputs of latches (strobed by ALE). Data valid for reads is measured at the data pins of the CPU. Data valid for writes is measured at the data inputs to the I/O device. IORC and IOWC are 82288 outputs. Figures 5-10 and 5-11 show I/O read and write timing for no-wait-state operation.

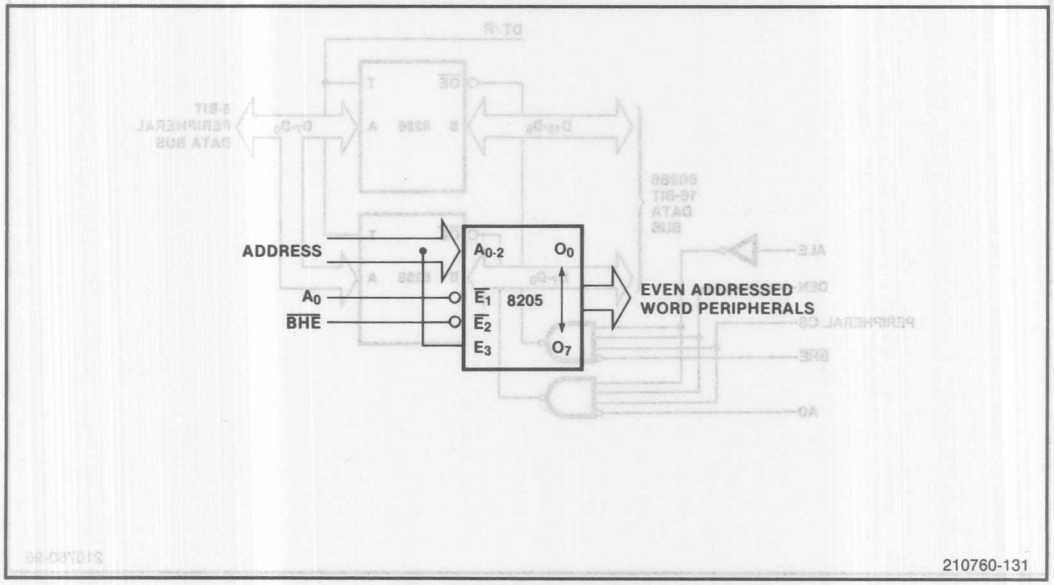


Figure 5-7. 16-bit I/O Decode

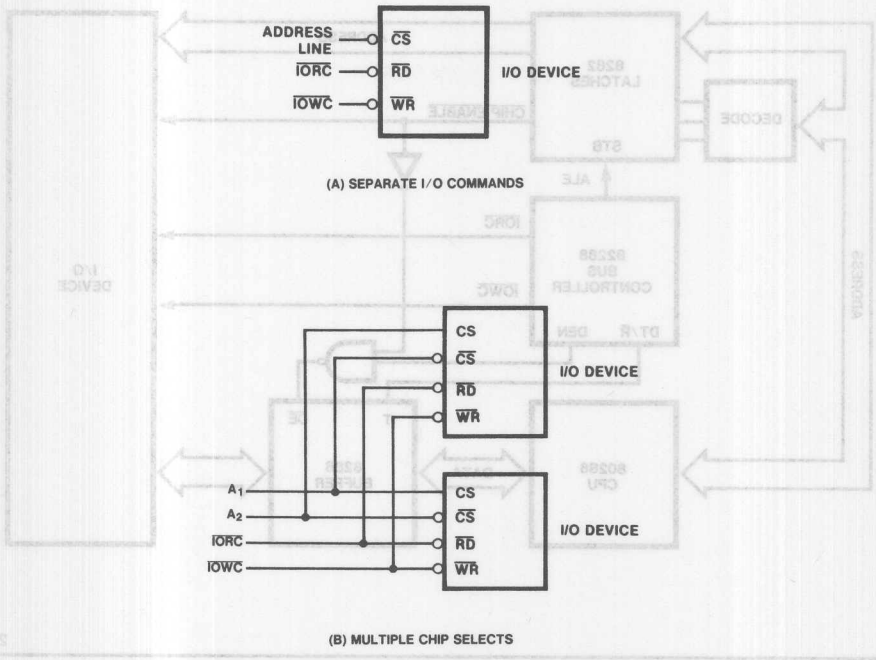


Figure 5-8. Linear Selects for I/O Devices

The timing parameters used in the analysis are defined as follows:

TLAVCML	=	Latched address valid to command active
TCMHLAX	=	Latched address hold from command inactive
TCMLCMH	=	Command pulse width
TCMLDV	=	Command active to data valid
TLAVDV	=	Latched address valid to data valid
TDVCMH	=	Data valid to command inactive
TCMHDX	=	Data hold from command inactive
TCMHDF	=	Data float from command inactive
TLCECML	=	Latched chip enable to command active
TCMHLCEX	=	Latched chip enable hold from command inactive
TLCEDV	=	Latched chip enable to data valid
TCMLCML	=	Interval from one operation to the next
TCMHCML	=	Interval between commands (not shown)

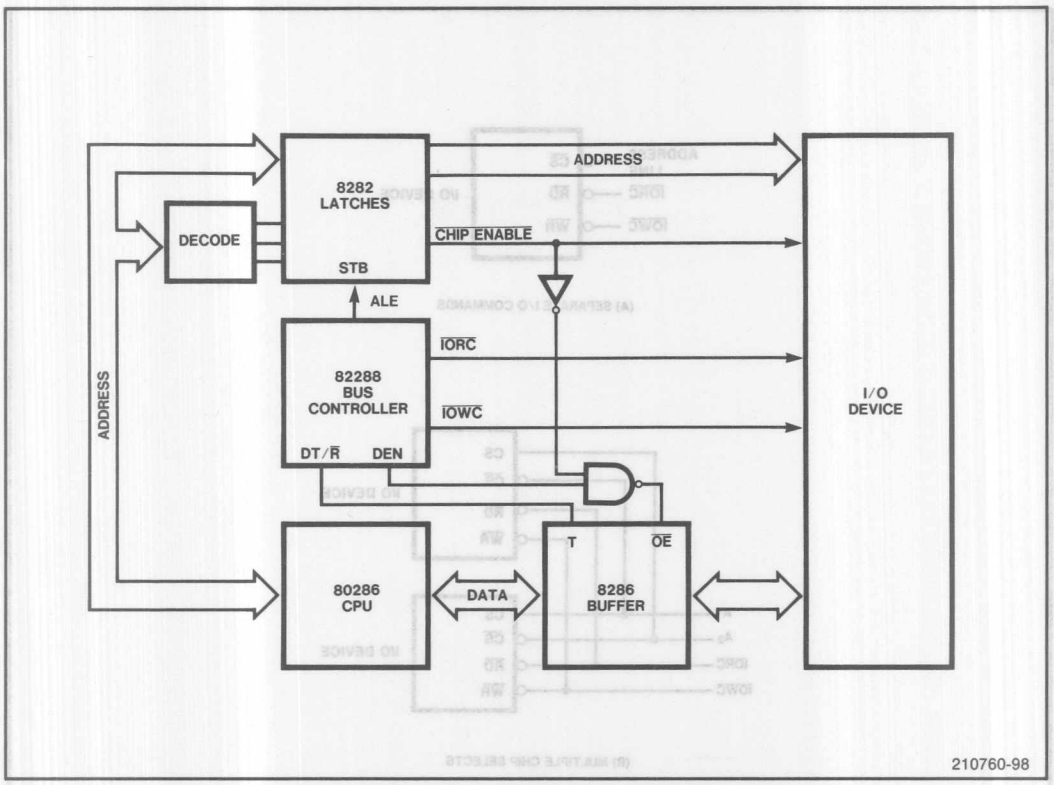


Figure 5-9. I/O Cycle Model

The worst-case timing values can be calculated by assuming the maximum delay in the latched address, chip select, and command signals, and the longest propagation delay through the data buffers (if present). These calculations provide the minimum possible access time and can be used to determine the compatibility of I/O devices.

Read Operations

For read operations, data must be valid at the pins of the 80286 10 ns before the falling edge of CLK at the end of T_c. The important timing parameters that determine whether I/O devices require one or more wait states are the address and command access times required by the device before read data will be valid.

Taking into consideration the maximum propagation delay through the 8286 data transceivers (30 ns max), the worst-case address access time provided during read operations with zero wait states is:

3 CLK cycles at 8 MHz	187.5 ns
− ALE active delay (max)	− 15.0 ns
− 8282 address latch delay (max)	− 45.0 ns
− 8286 data buffer delay (max)	− 30.0 ns
− Required data setup (min)	− 10.0 ns
Minimum address access time =	87.5 ns min

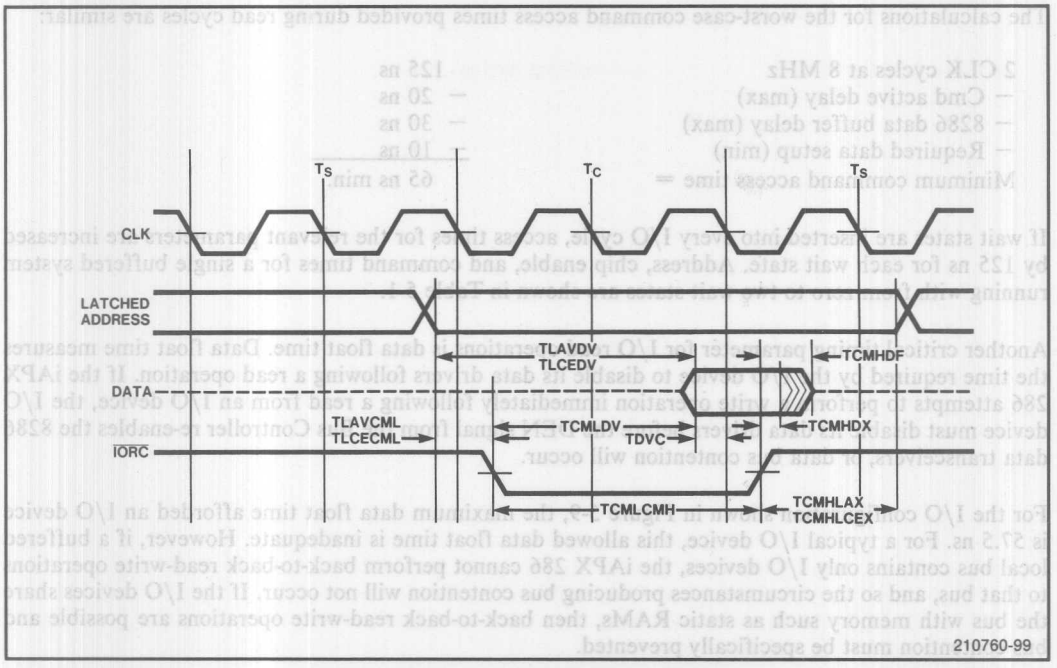


Figure 5-10. I/O Read Cycle Timing

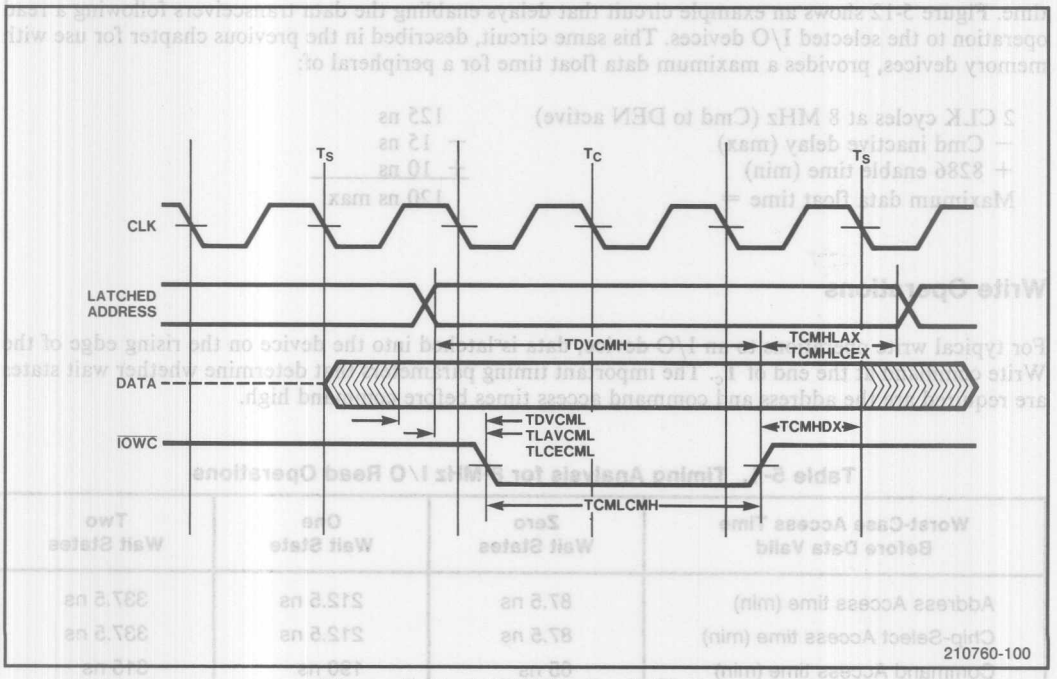


Figure 5-11. I/O Write Cycle Timing

The worst-case command access times provided during read cycles are similar:

2 CLK cycles at 8 MHz	125 ns
− Cmd active delay (max)	− 20 ns
− 8286 data buffer delay (max)	− 30 ns
− Required data setup (min)	− 10 ns
Minimum command access time =	65 ns min.

If wait states are inserted into every I/O cycle, access times for the relevant parameters are increased by 125 ns for each wait state. Address, chip enable, and command times for a single buffered system running with from zero to two wait states are shown in Table 5-1.

Another critical timing parameter for I/O read operations is data float time. Data float time measures the time required by the I/O device to disable its data drivers following a read operation. If the iAPX 286 attempts to perform a write operation immediately following a read from an I/O device, the I/O device must disable its data drivers before the DEN signal from the Bus Controller re-enables the 8286 data transceivers, or data bus contention will occur.

For the I/O configuration shown in Figure 5-9, the maximum data float time afforded an I/O device is 57.5 ns. For a typical I/O device, this allowed data float time is inadequate. However, if a buffered local bus contains only I/O devices, the iAPX 286 cannot perform back-to-back read-write operations to that bus, and so the circumstances producing bus contention will not occur. If the I/O devices share the bus with memory such as static RAMs, then back-to-back read-write operations are possible and bus contention must be specifically prevented.

To accommodate the data float time requirements of typical I/O devices, and thereby avoid bus contention, the output enables of the data bus tranceivers can be delayed for the necessary length of time. Figure 5-12 shows an example circuit that delays enabling the data transceivers following a read operation to the selected I/O devices. This same circuit, described in the previous chapter for use with memory devices, provides a maximum data float time for a peripheral of:

2 CLK cycles at 8 MHz (Cmd to DEN active)	125 ns
− Cmd inactive delay (max)	− 15 ns
+ 8286 enable time (min)	+ 10 ns
Maximum data float time =	120 ns max.

Write Operations

For typical write operations to an I/O device, data is latched into the device on the rising edge of the Write command at the end of T_c. The important timing parameters that determine whether wait states are required are the address and command access times before command high.

Table 5-1. Timing Analysis for 8-MHz I/O Read Operations

Worst-Case Access Time Before Data Valid	Zero Wait States	One Wait State	Two Wait States
Address Access time (min)	87.5 ns	212.5 ns	337.5 ns
Chip-Select Access time (min)	87.5 ns	212.5 ns	337.5 ns
Command Access time (min)	65 ns	190 ns	315 ns

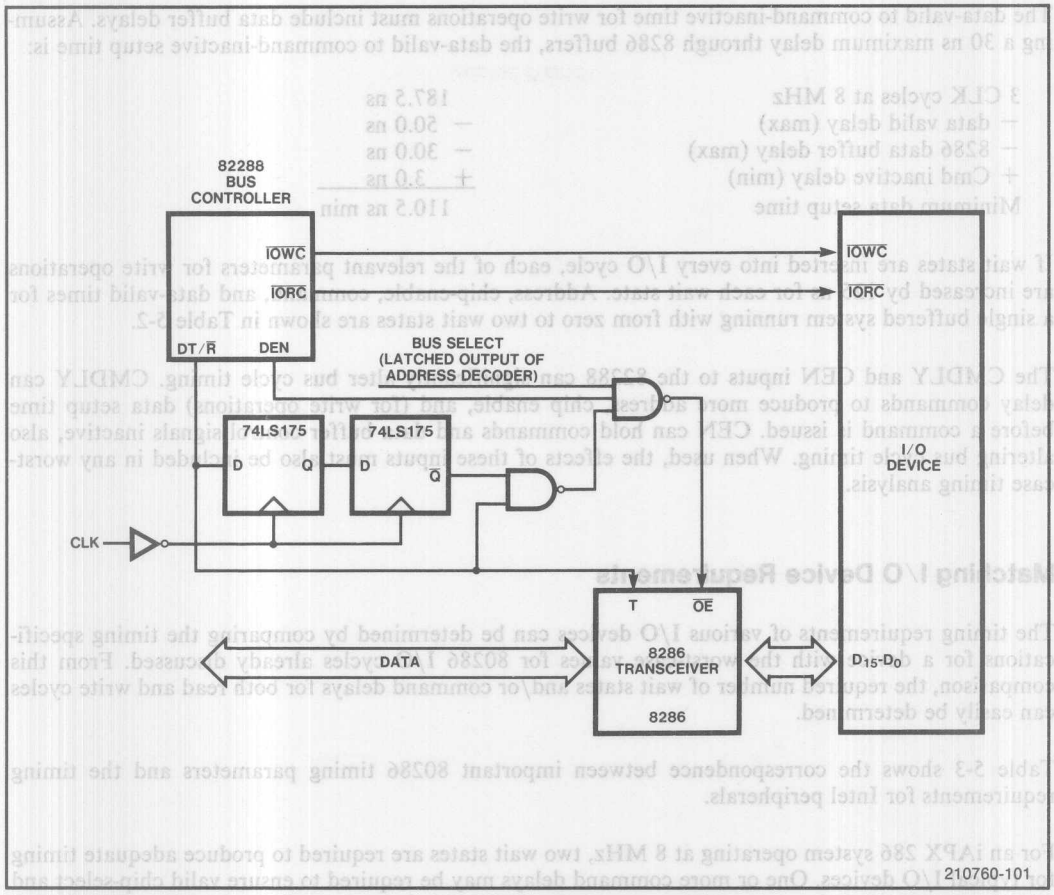


Figure 5-12. Delaying Transceiver Enable

The worst-case address-valid to command-high access time provided during write operations with zero wait states is:

3 CLK cycles at 8 MHz	187.5 ns
− ALE active delay (max)	− 15.0 ns
− 8282 address latch delay (max)	− 45.0 ns
+ Cmd inactive delay (min)	+ 3.0 ns
Minimum address access time =	130.5 ns min

Worst-case command-active to command-inactive timing is calculated in a similar manner (command timing for write operations is not affected by buffers in the data path):

2 CLK cycles at 8 MHz	125 ns
− Cmd active delay (max)	− 20 ns
+ Cmd inactive delay (min)	+ 3 ns
Minimum command access time =	108 ns min

The data-valid to command-inactive time for write operations must include data buffer delays. Assuming a 30 ns maximum delay through 8286 buffers, the data-valid to command-inactive setup time is:

3 CLK cycles at 8 MHz	187.5 ns
− data valid delay (max)	− 50.0 ns
− 8286 data buffer delay (max)	− 30.0 ns
+ Cmd inactive delay (min)	+ 3.0 ns
Minimum data setup time	110.5 ns min

If wait states are inserted into every I/O cycle, each of the relevant parameters for write operations are increased by 125 ns for each wait state. Address, chip-enable, command, and data-valid times for a single buffered system running with from zero to two wait states are shown in Table 5-2.

The CMDLY and CEN inputs to the 82288 can significantly alter bus cycle timing. CMDLY can delay commands to produce more address, chip enable, and (for write operations) data setup time before a command is issued. CEN can hold commands and data buffer control signals inactive, also altering bus cycle timing. When used, the effects of these inputs must also be included in any worst-case timing analysis.

Matching I/O Device Requirements

The timing requirements of various I/O devices can be determined by comparing the timing specifications for a device with the worst-case values for 80286 I/O cycles already discussed. From this comparison, the required number of wait states and/or command delays for both read and write cycles can easily be determined.

Table 5-3 shows the correspondence between important 80286 timing parameters and the timing requirements for Intel peripherals.

For an iAPX 286 system operating at 8 MHz, two wait states are required to produce adequate timing for typical I/O devices. One or more command delays may be required to ensure valid chip-select and address inputs before the command becomes active. Table 5-4 shows the wait-state and command-delay requirements of a variety of common peripherals.

I/O INTERFACE EXAMPLES

This section shows I/O interfaces to the 8274 Multi-Protocol Serial Controller, the 8255A-5 Programmable Peripheral Interface, and the 8259A-2 Programmable Interrupt Controller.

Table 5-2. 80286 Timing Analysis for I/O Write Operations

Worst-Case Access Time Before Command High	Zero Wait States	One Wait State	Two Wait States
Address Access time (min)	130.5 ns	255.5 ns	380.5 ns
Chip-Select Access time (min)	130.5 ns	255.5 ns	380.5 ns
Command Pulse width (min)	108 ns	233 ns	358 ns
Write Data Setup Time (min)	110.5 ns	235.5 ns	360.5 ns

Table 5-3. 80286/Peripheral Timing Parameters

80286 Cycle	Peripheral Read	Peripheral Write
TLAVCML	TAR	TAW
TCMHLAX	TRA	TWA
TCMLCMH	TRR	TWW
TCMLDV	TRD	—
TCMHCML	TDF	TRV
TLAVDV	TAD	—
TCMLCML	TRCYC	—
TDVCMH	—	TDW
TCMHDX	—	TWD
TLCECML	TAR	TAW
TCMHLCEX	TRA	TWA
TLCEDV	TRD	—

Table 5-4. Timing Requirements for Selected Peripherals

Intel Peripheral	Required Wait States	Required Command Delays
8251A	2	1
8254-2	2	1
8255-5	2	0
8259A	2	0
8271	2	0
8272	2	0
8274	2	0
8291	2	0
8295A	2	0

8274 Interface

The 8274 Multi-Protocol Serial Controller (MPSC) is designed to interface high-speed serial communications lines using a variety of communications protocols, including asynchronous, IBM bi-synchronous, and HDLC/SDLC protocols. The 8274 contains two independent full-duplex channels, and can serve as a high-performance replacement for two 8251A Universal Synchronous/Asynchronous Receiver Transmitters (USARTs).

Figure 5-13 shows the signals required to interface an 8274 MPSC to an iAPX 286. The 8274 MPSC is accessed by the iAPX 286 as a sequence of four 8-bit I/O-address or memory-address locations. For interrupt operation, the 8274 can respond to 80286 interrupt-acknowledge sequences in the same manner as an 8259A Interrupt Controller, and can be used in conjunction with a master 8259A Interrupt Controller in a cascaded configuration.

The chip-select and address inputs (CS, A0, and A1) to the 8274 must all be latched. Typically, address input A0 and A1 would be connected to the latched address lines A1 and A2, respectively, of the local address bus. A single level of buffering is typically used on the 8 data lines between the CPU and the 8274 MPSC.

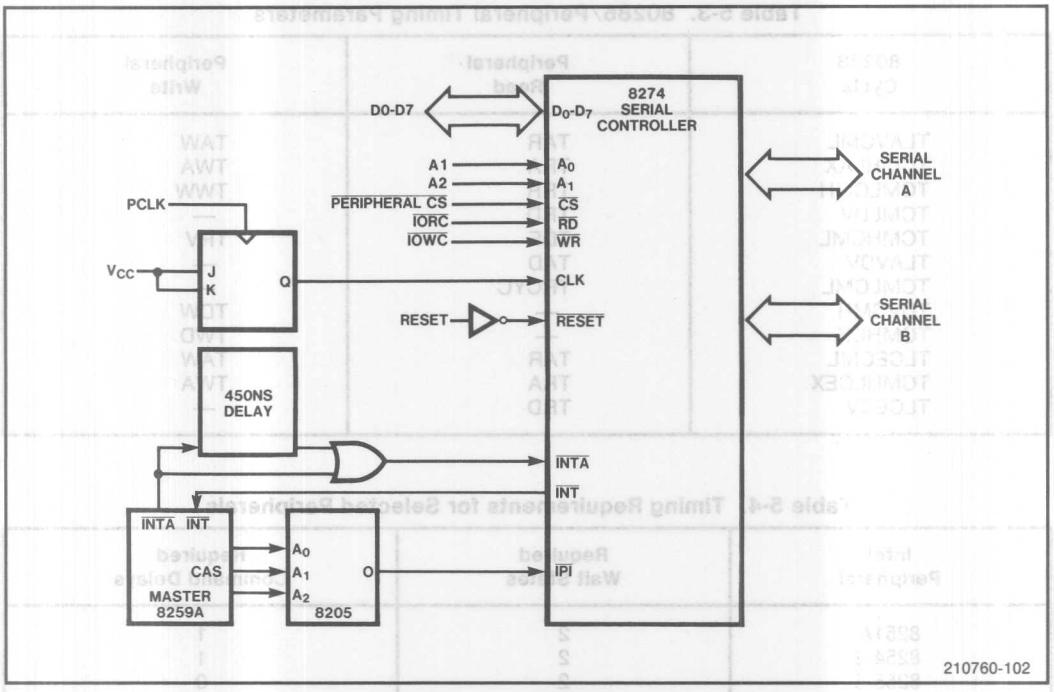


Figure 5-13. 8274 MPSC Interface

The 8274 $\overline{RD}$ and $\overline{WR}$ commands are connected to the $\overline{IORC}$ and $\overline{IOWC}$ outputs of the 82288 Bus Controller. To provide an acceptable CLK input for the 8274 MPSC, the 82284 PCLK clock frequency can be divided by two and used to drive the 8274 CLK input.

For Interrupt operation, the $\overline{INTA}$ signal from the Bus Controller is also connected to the 8274 MPSC. When using the 8274 in a cascaded interrupt configuration, the $\overline{IPI}$ input from the cascade address decoder must be valid before the falling edge of the $\overline{INTA}$ signal. A 450 ns delay circuit is used to delay $\overline{INTA}$ until this decoded CAS address becomes valid.

Further details on interfacing the 8274 MPSC to a microprocessor system can be found in the appropriate data books. The following discussion looks at the bus timing requirements for accessing the 8724 MPSC on a buffered iAPX 286 local bus.

READ TIMING

The Read timing requirements for the 8274 MPSC are as follows:

TARmin = 0 ns
TRAmin = 0 ns
TRRmin = 250 ns
TRDmax = 200 ns
TDFmax = 120 ns

A comparison of TAR (0 ns min. required) with 80286 timing (5.5 ns min. provided) shows that no command delays are required. The command pulse width requirement (TRRmin) of 250 ns minimum indicates that at least two wait states must be inserted into the iAPX 286 bus cycle (command pulse width with two wait states is 310 ns). With these two wait states, read operations to the 8274 result in a data access time of at least 315.5 ns, easily meeting the 8274 requirement for at least 200 ns.

Data float time for the 8274 MPSC is 120 ns maximum, and may necessitate the delayed-transceiver-enable circuit shown in Figure 5-12. The circumstances requiring a delayed transceiver-enable to accommodate lengthy data float times have been described in the previous section on Read Operations.

WRITE TIMING

Write timing requirements for the 8274 MPSC are as follows:

TAWmin = 0 ns
TWAmin = 0 ns
TWWmin = 250 ns
TDWmin = 150 ns
TWDmin = 0 ns
TRVmin = 300 ns

The first three timing requirements (TAWmin, TWAmin, and TWWmin) are identical to their equivalent read parameters, and require identical timing. Like Read operations, write operations to the 8274 must have two wait states inserted into the iAPX 286 bus cycle.

The remaining three parameters are specific to write cycles. With two wait states, the iAPX 286 easily exceeds the minimum 8274 data setup time (TDWmin = 150 ns) by providing at least 255.5 ns. The 8274 requires no data hold time.

TRVmin is the recovery time between write cycles to the USART. Recovery time is required following any mode changes or control accesses. The circuit shown, however, guarantees only 62.5 ns between successive Write commands. The proper recovery time between successive write cycles can be obtained easily through software delays, or can be implemented in hardware by delaying commands by an appropriate four CLK cycles.

8255A-5 Interface

Figure 5-14 shows the interface between an 8255A-5 Programmable Peripheral Interface and an iAPX 286. Timing parameters are as follows:

Read Timing		Write Timing	
TARmin	= 0 ns	TAWmin	= 0 ns
TRAmin	= 0 ns	TWAmin	= 20 ns
TRRmin	= 300 ns	TWWmin	= 300 ns
TRDmax	= 200 ns	TDWmin	= 100 ns
TDFmax	= 100 ns	TWDmin	= 30 ns

TCYCmin (Reads and Writes) = 850 ns

A bus cycle with two wait states and no command delay, and using the additional delayed-transceiver-enable logic to delay enabling the data transceivers following a read operation, meets all of the timing requirements except for the time between cycles (TCYCmin). This remaining parameter can be met

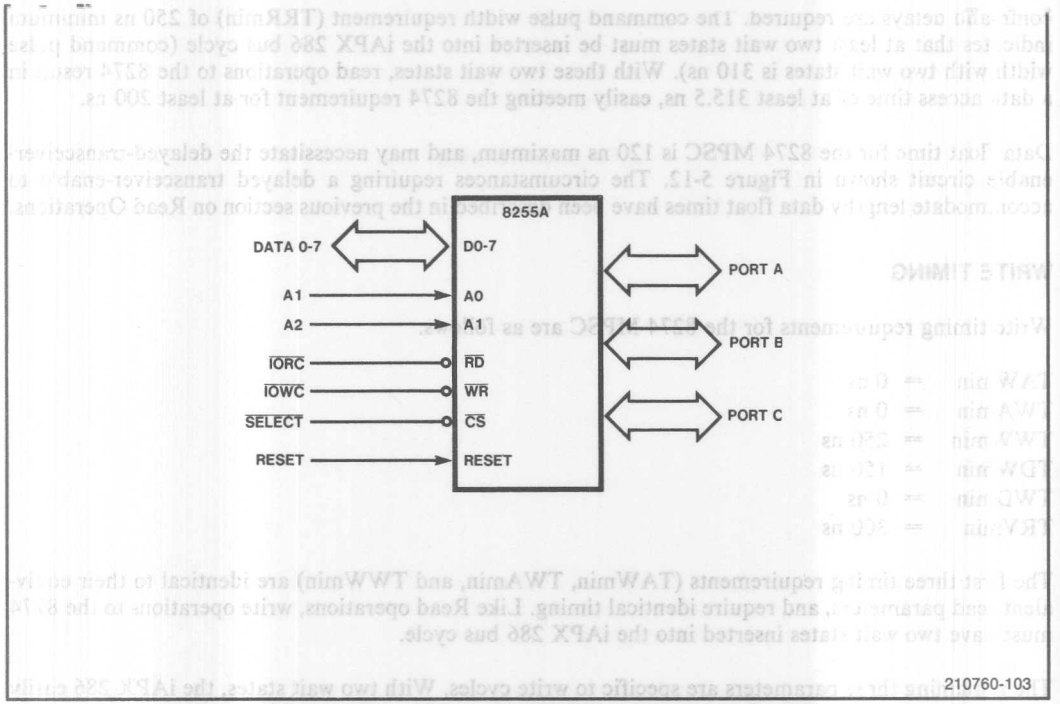


Figure 5-14. 8255A-5 Parallel Port Interface

through an appropriate software delay or in hardware by delaying commands the required number of CLK cycles (and inserting wait states to meet the other requirements).

8259A-2 Interface

The 8259A-2 Programmable Interrupt Controller is designed for use in interrupt-driven microcomputer systems, where it manages up to eight independent interrupt sources. The 8259A-2 handles interrupt priority-resolution and individual interrupt masking, and directly supports the iAPX 286 manner of acknowledging interrupts. During iAPX 286 interrupt-acknowledge sequences, the 8259A resolves the highest-priority interrupt that is currently active, and returns a pre-programmed interrupt vector to the 80286 to identify the source of the interrupt.

A single 8259A-2 Interrupt Controller can handle up to eight external interrupts. Multiple 8259A-2 Interrupt Controllers can be cascaded to accommodate up to 64 interrupt requests. A technique for handling more than 64 external interrupts is discussed at the end of this section.

Intel Application Note AP-59 contains much more detailed information on configuring an 8259A in a variety of different ways. The remainder of this section contains specific details for interfacing an 8259A Interrupt Controller to the iAPX 286.

SINGLE INTERRUPT CONTROLLER

Figure 5-15 shows the interface between the 80286 CPU and a single 8259A-2 Interrupt Controller. Timing parameters for the 8259A-2 are as follows (note that symbols used in the 8259A data sheet differ from those used for most of the other Intel peripherals):

Symbol	Parameter	ns
TAHRLmin	A0/CS Setup to RD/INTA Active	0
TRHAXmin	A0/CE Hold from RD/INTA Inactive	0
TRLRHmin	RD/INTA Pulse Width	160
TRLDVmax	Data Valid from RD/INTA Active	120
TRHDZmax	Data Float from RD/INTA Inactive	100
TRHRLmin	End of RD to next RD or End of INTA to next INTA within an INTA sequence only	160
TAHWLmin	A0/CS Setup to WR Active	0
TWHAXmin	A0/CS Hold from WR Inactive	0
TWLWHmin	WR Pulse Width	190
TDVWHmin	Data Valid to WR Inactive	160
TWHDXmin	Data Hold from WR Inactive	0
TWHWLmin	End of WR to Next WR	190
TCHCLmin	End of Command to Next Command (not same type) or End of INTA Sequence to Next INTA Sequence	500

To ensure proper operation, bus operations accessing the 8259A require one wait state. No CMDLYs are necessary. Because the 8259A requires a minimum of 500 ns (typically, two bus cycles) between successive reads or writes, software accessing the 8259A should be suitably written to avoid violating this requirement.

When an interrupt occurs, the 80286 CPU automatically executes two back-to-back interrupt-acknowledge bus cycles. The timing of these interrupt-acknowledge (INTA) cycles is described in Chapter Three. The external Ready logic must insert at least one wait state into each INTA cycle to ensure proper 8259A timing. No CMDLYs are necessary. Between INTA cycles, the 80286 automatically inserts three idle (T_i) cycles in order to meet 8259A timing requirements.

CASCADED INTERRUPT CONTROLLERS

Figure 5-16 shows the interface between the 80286 CPU and multiple 8259A-2 interrupt controllers. The master Interrupt Controller resides on the local bus, while up to eight slave controllers can be interfaced to the system bus. Slave controllers resolve priority between up to eight interrupt requests and transmit single interrupt requests to the master controller. The master controller, in turn, resolves interrupt priority between up to eight slave controllers and transmits a single interrupt request to the 80286 CPU. Up to 64 interrupt requests can be accommodated by the configuration shown.

The basic read/write timing for cascaded interrupt controllers is the same as for a single interrupt controller. The timing for interrupt-acknowledge sequences, however, involves additional signals over those required for a single interrupt subsystem.

During the first interrupt-acknowledge cycle, the master controller and all slave controllers freeze the state of their interrupt request inputs. The master controller outputs a cascade address that is enabled by MCE (Master Cascade Enable) from the 82288 and latched by ALE; the cascade address is typically gated onto address lines A8-A10 (IEEE 796 Multibus standard). This cascade address selects the slave

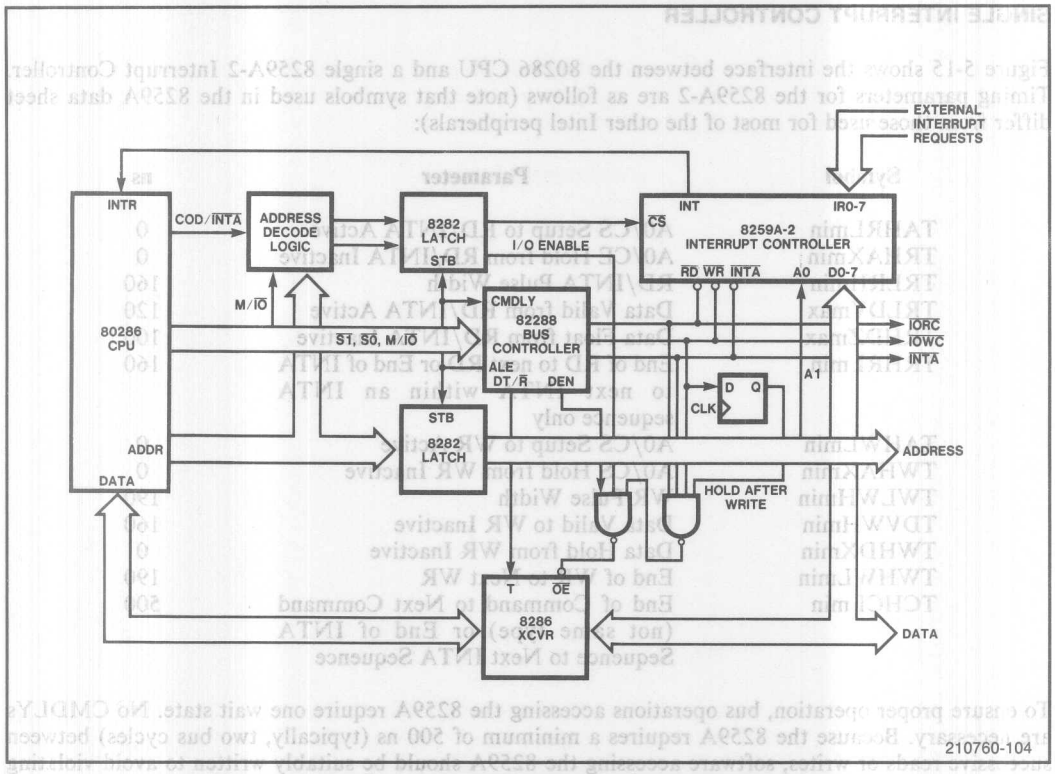


Figure 5-15. Single 8259A Interrupt Controller Interface

controller that is generating the highest-priority interrupt request. During the second interrupt-acknowledge bus cycle, the slave controller responding to the cascade address outputs an interrupt vector that points to an appropriate interrupt service routine.

Chapter Seven includes details for designing systems using cascaded Interrupt Controllers when slave controllers may reside on a system bus such as the Multibus.

HANDLING MORE THAN 64 INTERRUPTS

Cascaded 8259A Interrupt Controllers can accommodate up to 64 independent interrupt requests. iAPX 286 systems that require more than 64 interrupts can use additional 8259A-2 devices in a polled mode.

For example, interrupt request inputs to a slave controller can be driven by a third level of 8259A-2 controllers. When one of these additional controllers receives an interrupt request, it drives one of the interrupt request inputs to the slave controller active. The slave controller signals the master controller, which in turn interrupts the CPU. The interrupt identifier received from the slave controller directs the CPU to a service routine that polls the third level of interrupt controllers to determine the source of the request.

The only additional hardware required to handle more than 64 interrupt sources are the additional 8259A-2 devices and address-decode logic for selecting the additional devices. Polling is performed by writing the poll command to the appropriate 8259A-2, followed immediately by a read command. For

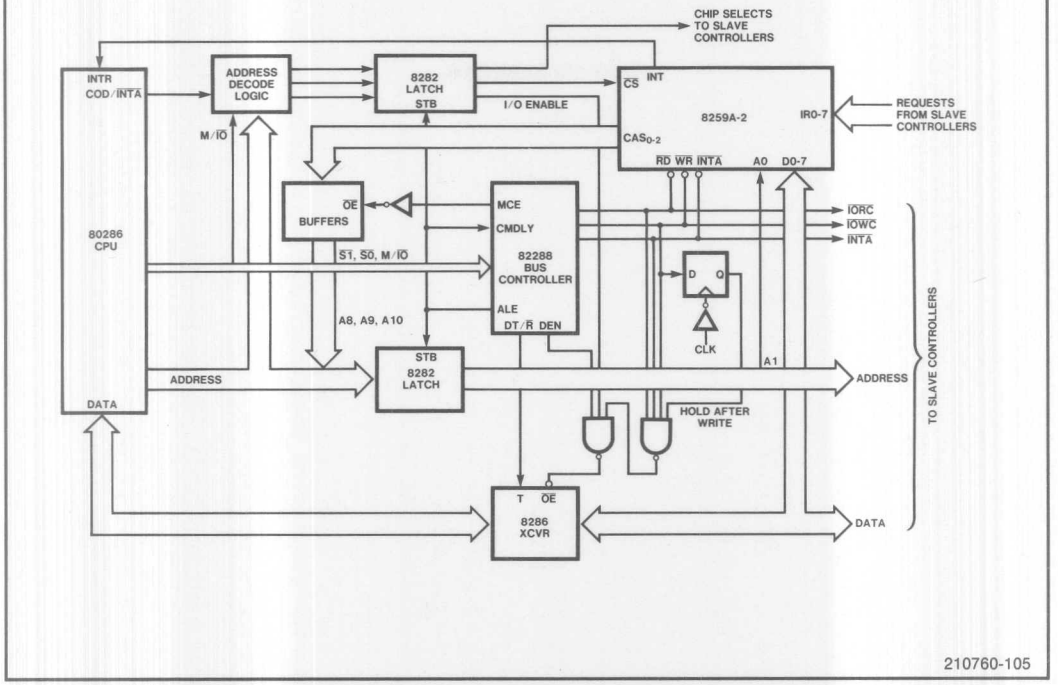


Figure 5-16. Cascaded 8259A Interrupt Controller Interface

maximum performance, use of a third level of interrupt controllers should be restricted to less-critical, less frequently-used interrupts.

THE iSBX BUS—A MODULAR I/O EXPANSION BUS

The iSBX Bus is a modular I/O expansion bus that allows single-board computer systems to be expanded simply by plugging in specialized iSBX Multimodule boards.

The iSBX Multimodule boards respond to fully-decoded chip select lines defined on the bus. An $\overline{\text{MPST}}$ control signal indicates to the computer system that a Multimodule board is present. Once a Multimodule board has been installed, the I/O devices on the Multimodule board appear as an integral part of the single-board computer, and can be accessed in software using the standard I/O instruction set.

The timing of bus operations between an iAPX 286 and an iSBX Multimodule Board is controlled by the iSBX $\overline{\text{MWAIT}}$ signal. This signal is asserted to insert wait states with the 80286 bus cycle, and can simply be inverted and used to drive the 82284 $\overline{\text{ARDY}}$ input.

The iSBX Bus is described in the *iSBX Bus Specification*, Order Number 142686-002. This document contains complete details for designing Multimodule boards compatible with the iSBX bus standard.

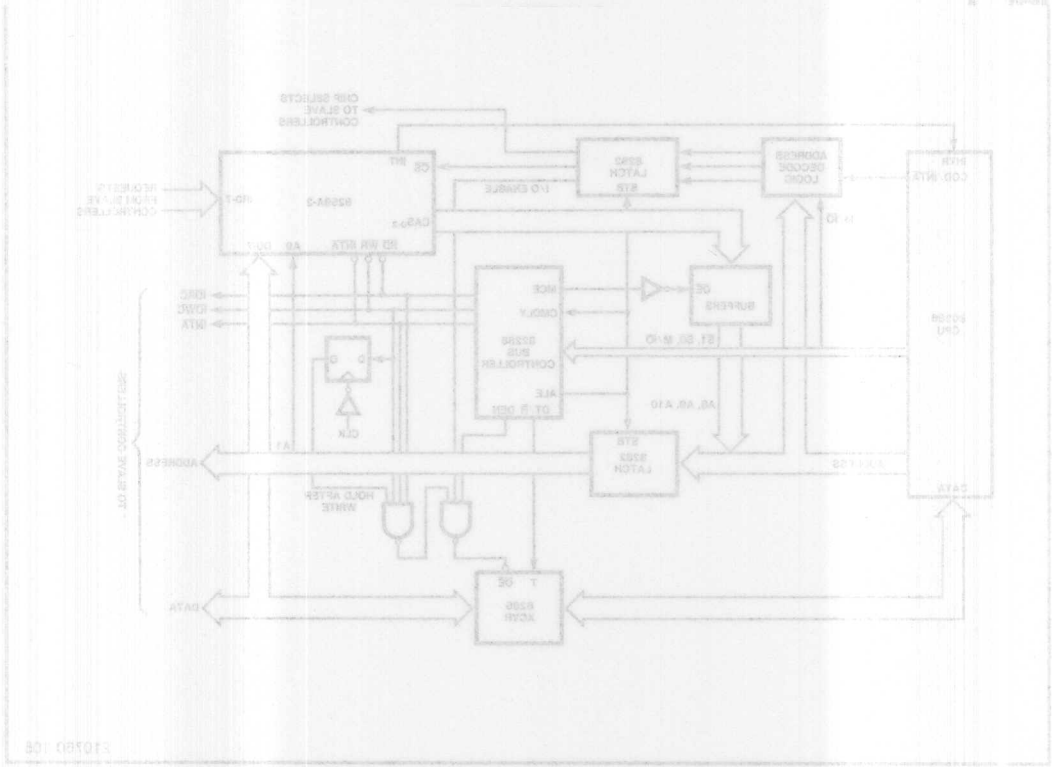


Figure 5-16. Cascaded 8259A Interrupt Controller Interface

Using The 80287 Numeric Processor Extension

6

6 Using The 80287 Numeric Processor Extension

CHAPTER 6
USING THE 80287 NUMERIC PROCESSOR EXTENSION

The Intel 80287 is a high-performance numeric processor extension that extends the iAPX 286/10 system by adding floating-point, extended-integer, and BCD data types. The iAPX 286/20 system, comprising an 80286 processor with an 80287 processor extension, contains over fifty additional instructions over those of an iAPX 286/10, and fully conforms to the proposed IEEE 754 Floating Point Standard. The additional instructions added by the Numeric Processor Extension are described in the 8086 Numeric Supplement as well as in the 80287 Data Sheet.

This chapter details how to design an iAPX 286/20 system connecting the 80287 Numeric Processor to an iAPX 286/10.

- The first section of this chapter describes the electrical connections of the 80287 to an iAPX 286/10 system.
- The second section describes the local bus activity you may observe using the 80287 with the 80286. This bus activity includes:

 A. Interactions between the 80286 and 80287 under program control (executing ESC instructions).

 B. Interactions that occur asynchronous to the program, where the 80287 Numeric Processor requests the transfer of operands between itself and system memory using the 80286 Processor Extension Data Channel.

- The final section of this chapter describes how to design an upgradable iAPX 286/10 system with an empty 80287 socket. This system may be upgraded to an iAPX 286/20 system simply by inserting an 80287 into the empty socket. This section includes an example software routine to recognize the presence of an 80287.

THE 80287 PROCESSOR EXTENSION INTERFACE

The 80287 can be connected to an iAPX 286/10 system as shown in Figure 6-1.

Four important points should be observed when connecting the 80287 to an iAPX 286/10 system:

1. The 80287 operates as an extension of the 80286: the 80287 connects directly to the status lines of the 80286. The 80286 executes programs in the normal manner; the 80287 automatically executes any numeric instructions when they are encountered.
2. The 80287 responds to particular I/O addresses (00F8H, 00FAH, and 00FCH) automatically generated by the 80286.
3. The 80287 can be driven by a separate clock signal, independent of the 80286 clock. This allows a higher-performance (8 MHz) 80287 to be used if system performance requirements warrant it.
4. Because the 80287 data lines are connected directly to those of the 80286, the buffer/drivers driving the local data bus must be disabled when the 80286 reads from the 80287.

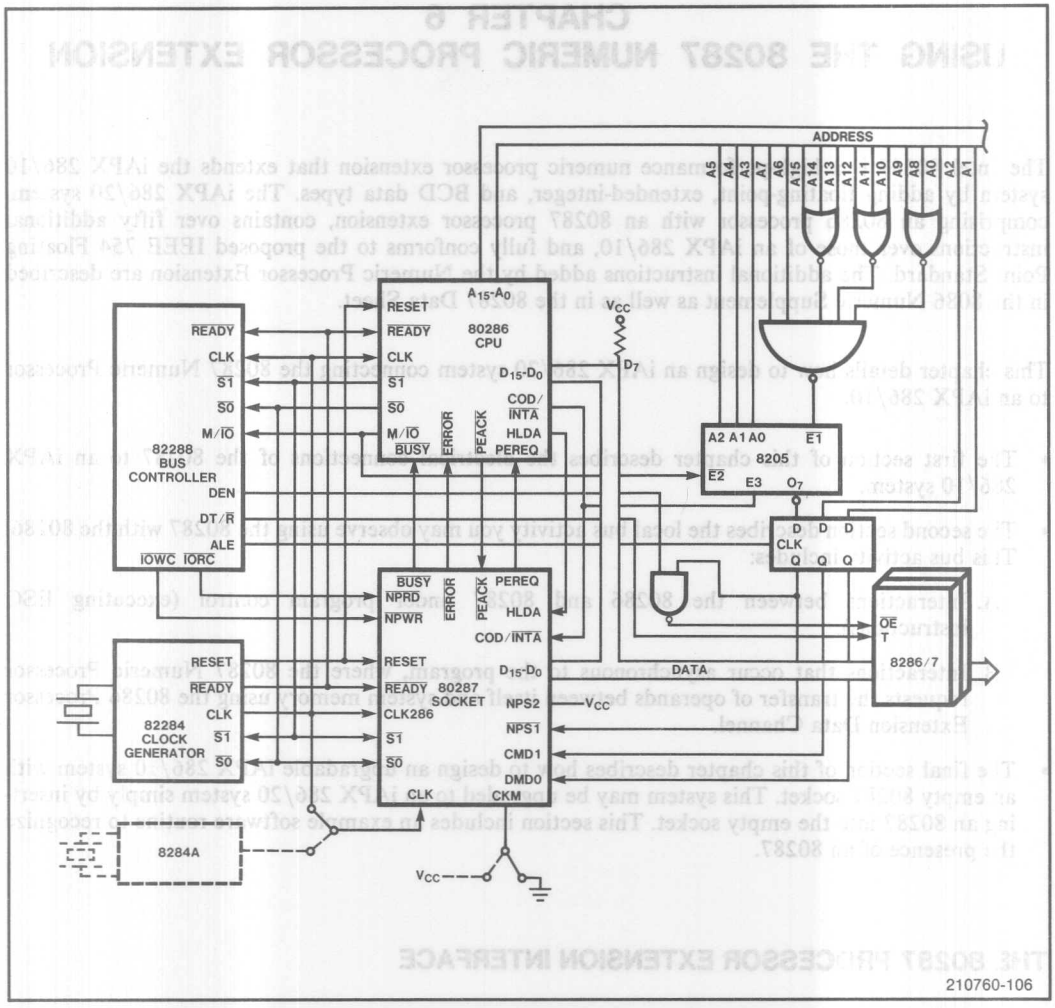


Figure 6-1. iAPX 286/20 System Configuration

The 80287 Status Lines

The 80287 has a number of status lines and inputs that are connected directly to the 80286.

The $\overline{S1}$, $\overline{S0}$, COD/$\overline{INTA}$, $\overline{READY}$, RESET, HLDA, and CLK pins of the 80287 are connected to the corresponding pins of the 80286. By monitoring these signals, the 80287 can observe the execution of ESC instructions by the 80286.

The $\overline{BUSY}$ signal from the 80287 is connected directly to the 80286; it signals that the Processor Extension is currently executing a numeric instruction. The 80286 will not execute most ESC instructions until this $\overline{BUSY}$ signal becomes inactive. The 80286 WAIT instruction and most ESC instructions cause the 80286 to wait specifically until this signal becomes inactive.

The $\overline{\text{ERROR}}$ signal from the 80287 is also connected directly to the 80286; it signals that the previous numeric instruction caused an unmasked exception condition. The 80287 Data Sheet describes these exception conditions and explains how these exceptions may be masked under program control. If an exception occurs, this $\overline{\text{ERROR}}$ signal becomes active before the $\overline{\text{BUSY}}$ signal goes inactive, signalling the end of the numeric instruction.

Addressing the 80287

When the 80286 executes an ESC instruction, the 80286 automatically generates one or more I/O operations to the 80287's reserved I/O addresses. These I/O operations take place independent of the 80286's current I/O privilege level.

Table 6-1 shows the particular I/O addresses reserved for the 80287 and shows how the four processor-select and command inputs of the 80287 must be activated when these I/O addresses are asserted. The CMD0 and CMD1 signals of the 80287 may be connected to the latched A1 and A2 address lines, and, in addition, the $\overline{\text{NPRD}}$ and $\overline{\text{NPWR}}$ signals of the 80287 should be connected to the $\overline{\text{IORC}}$ and $\overline{\text{IOWC}}$ signals from the 82288 Bus controller, respectively.

The 80287 Clock Input

The 80287 can operate either directly from the CPU clock or with a dedicated clock. To operate the 80287 from the CPU clock, the CKM pin of the 80287 is tied to ground. In this mode, the 80287 internally divides the system clock frequency to operate at one-third the frequency of the system clock (i.e., for an 8 MHz 80286, the 16 MHz system clock is internally divided down to 5.3 MHz).

To use a higher-performance (8 MHz) 80287, the CKM pin of the 80287 must be tied high, and an 8284A clock driver and appropriate crystal may be used to drive the 80287 with an 8 MHz, 33% duty-cycle, MOS-level clock signal on the CLK input. In this mode, the 80287 does not internally divide the clock frequency; the 80287 operates directly from the external clock.

Table 6-1. I/O Address Decoding for the 80287

I/O Address (Hexadecimal)	80287 Select and Command Inputs			
	NPS2	$\overline{\text{NPS1}}$	CMD1	CMD0
00F8	1	0	0	0
00FA	1	0	0	1
00FC	1	0	1	0
00FE	* Reserved For Future Use			

NOTE:
These addresses are generated automatically by the 80286. Users should not attempt to reference these I/O addresses explicitly, at the risk of corrupting data within the 80287.

The 80287 operates as a parallel processor independent of the 80286, and interacts with the 80286 in two distinct ways:

1. The 80286 initiates 80287 operations during the execution of an ESC instruction. These interactions occur under program control; thus, they are easily recognized as part of the instruction stream.
2. The 80287 requests the 80286 to initiate operand transfers using the Processor Extension Data Channel of the 80286. These operand transfers between the 80287 and system memory occur when the 80287 requests them; thus, they are asynchronous to the regular instruction stream of the 80286.

Execution of ESC Instructions

When the 80286 encounters an ESC instruction, the 80286 first checks for the presence of the Processor Extension and verifies that the Processor Extension is in the proper context. If the BUSY status line from the 80287 is active, the 80286 waits for this signal to become inactive before proceeding to execute the ESC instruction.

When the 80286 executes an ESC instruction, the 80286 automatically generates one or more I/O operations to the 80287's reserved I/O addresses. These I/O operations take place independent of the 80286's current I/O Privilege level or Current Privilege level. The timing of these I/O operations is similar to the timing of any other I/O operation, with no (zero) wait states required for successful transfers.

Figure 6-2 illustrates the timing of data transfers with the 80287.

The Processor Extension Data Channel

All transfers of operands between the 80287 and system memory are performed by the 80286's internal Processor Extension Data Channel. This independent, DMA-like data channel permits all operand transfers of the 80287 to fall under the supervision of the 80286 memory-management and protection model.

DATA CHANNEL REQUESTS

When the 80286 executes an ESC instruction that requires transfers of operands either to or from the 80287, the 80286 automatically initializes the Processor Extension Data Channel, setting the memory address base and memory address limit registers, and setting the direction flag to indicate the direction of the transfer. Once the 80826 has initialized the Processor Extension Data Channel, the Processor Extension can request operand transfers through the Data Channel by raising PEREQ active. This request line remains high until the 80286 acknowledges the request by lowering its $\overline{\text{PEACK}}$ signal.

Figure 6-3 illustrates the timing of PEREQ and $\overline{\text{PEACK}}$ in controlling the operation of a Data Channel transfer. $\overline{\text{PEACK}}$ always goes active during the first bus operation of a Data Channel transfer.

DATA CHANNEL TRANSFERS

Numeric data transfers performed by the Processor Extension Data Channel use the same timing as any other 80286 bus cycle. Figure 6-2 illustrates operand transfers between the 80287 and system memory, placing the 16-bit operands at even-aligned word boundaries.

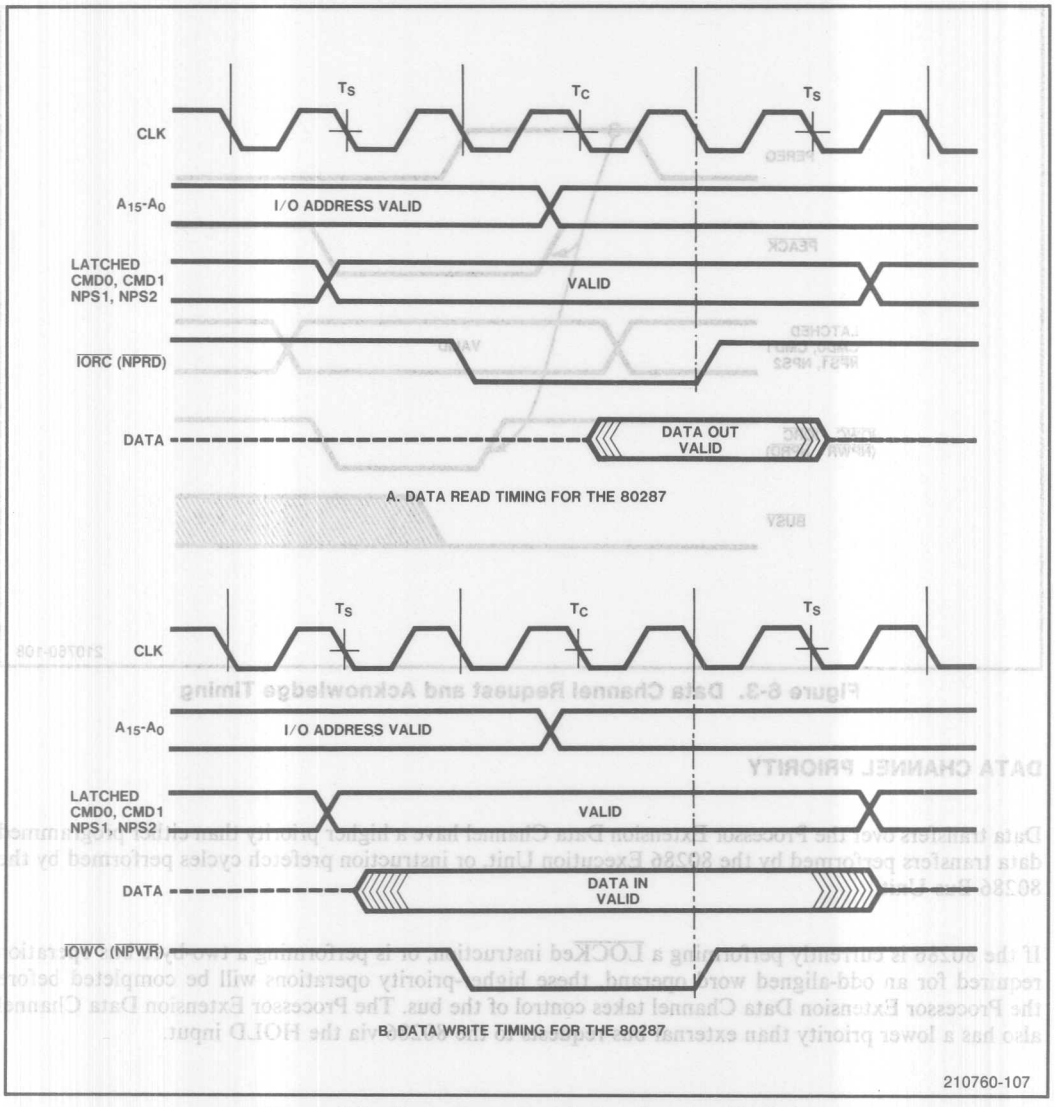


Figure 6-2. Data Transfer Timing for the 80287

For each operand transfer over the Processor Extension Data Channel, two or three bus operations are performed: one (I/O) bus operation to the 80287, and one or two bus operations to transfer the operand between the 80286 and system memory. Normally, Data Channel transfers require only two bus cycles; three bus cycles are required for each operand aligned on an odd byte address. The timing of word transfers to odd-aligned addresses is described in Chapter Three.

Operand transfers over the Processor Extension Data Channel may occur at any time following an ESC instruction as long as the Processor Extension's $\overline{\text{BUSY}}$ signal is active. Once this $\overline{\text{BUSY}}$ signal becomes inactive, no further requests by the 80287 for the Data Channel will occur until after the execution of a subsequent ESC instruction.

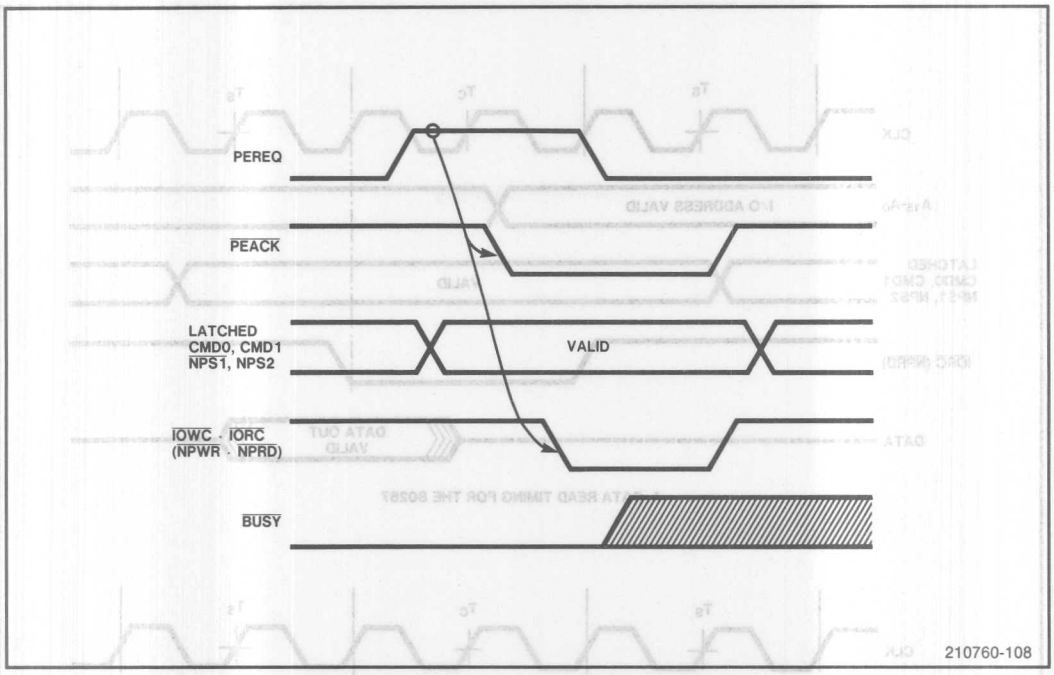


Figure 6-3. Data Channel Request and Acknowledge Timing

DATA CHANNEL PRIORITY

Data transfers over the Processor Extension Data Channel have a higher priority than either programmed data transfers performed by the 80286 Execution Unit, or instruction prefetch cycles performed by the 80286 Bus Unit.

If the 80286 is currently performing a $\overline{\text{LOCK}}$ed instruction, or is performing a two-byte bus operation required for an odd-aligned word operand, these higher-priority operations will be completed before the Processor Extension Data Channel takes control of the bus. The Processor Extension Data Channel also has a lower priority than external bus requests to the 80286 via the HOLD input.

Performance Using 80287 Parallel Processing

Because the 80287 Numeric Processor executes instructions in parallel with the 80286, the performance of the iAPX 286/20 system is fairly insensitive to memory speed and the number of wait states encountered by the 80286.

Table 6-2 illustrates how the performance of an iAPX 286/20 system is only slightly degraded as the number of wait states is increased. The figures shown are for an 8 MHz 80286-A1 with a 5 MHz 80287-A1 operating in Real mode. For an explanation of the Double-Precision Whetstone Benchmark used in this test, see "A Synthetic Benchmark," H. J. Curnow and B. A. Wichmann; *Computer Journal*, Volume 19, No. 1.

Table 6-2. Whetstone Performance with Multiple Wait States

Number of Wait States	Performance	
	dWhets*	Relative to 0-wait states
0	147.9	1.00
1	142.5	0.96
2	138.5	0.94
3	135.4	0.92

*Performance shown in thousands of double-precision Whetstone instructions per second.

DESIGNING AN UPGRADABLE iAPX 286 SYSTEM

When designing an iAPX 286/10 system, it is relatively easy to design a socket for the 80287 Numeric Processor Extension that permits the system to be upgraded later to an iAPX 286/20 system. Upgrading the system at a later date is accomplished simply by installing the 80287 Numeric Processor in its socket—no switches or strapping options are required. Using an appropriate initialization sequence, the iAPX 286 system can determine whether an 80287 is present in the system, and respond accordingly. This capability allows a single design to address two different levels of system price vs. performance.

Designing the 80287 Socket

Figure 6-1 shown previously illustrates one way to design an 80287 socket to allow upgradability. A single pull-up resistor is attached to data line D_7 to ensure that this data bit will read high during a floating condition. As explained in the following section, this data bit reading high will properly indicate the absence or presence of the 80287 Numeric Processor.

Recognizing the 80287

During initialization, the 80286 can be programmed to recognize the presence of the 80287 Numeric Processor Extension. Figure 6-4 shows an example of such a recognition routine.

In the example routine, the 80286 assumes that the 80287 is present, and executes an FNINIT instruction. Following the FNINIT instruction, the 80286 reads the 80287 status word. If an 80287 is indeed present, the lower 8-bits of this word (the exception flags) will all be zeroes. If an 80287 is not present, these data lines will have been floating. The preceding section of this manual explained how to design the 80287 socket to ensure that at least one of these lower-eight data lines floats high in the absence of the 80287.

Depending on the contents of these data lines, the 80286 can determine the presence or absence of an 80287 and then accordingly set or clear the MP and EM bits in its own Machine Status Word. If the 80286 determines that the 80287 has been installed, the 80286 will automatically pass all subsequent numeric instructions to the 80287 for execution. If the 80287 is not present, the 80286 can automatically invoke a software emulation routine each time it encounters a numeric instruction in its instruction path.

```
; initialization routine to detect an 80287 Numeric Processor

FND_287: FNINIT          ; initialize Numeric Processor
         FSTSW AX        ; retrieve 80287 status word
         OR    AL,AL     ; test low-byte--80287 exception flags
                         ; if all zero, then 80287 present and
                         ; properly initialized
                         ; if not all zero, then 80287 absent.
         JZ    GOT_287   ; branch if 80287 present

         SMSW  AX        ; No Numeric Processor--
         OR    04H       ; set EM bit in machine status word
         LMSW  AX        ; to enable software emulation of 80287
         JMP   CONTINUE

GOT_287: SMSW  AX        ; Numeric Processor present
         OR    02H       ; set MP bit in machine status word
         LMSW  AX        ; to permit normal 80287 operation
CONTINUE:                ; and off we go ...
```

Figure 6-4. Software Routine to Recognize the 80287

The System Bus 7

7 The System Bus

CHAPTER 7
THE SYSTEM BUS

The concept of a system bus has already been introduced in Chapter Two. A system bus connects one or more processing elements, each of which shares access to the same system resources. One or all of the processing elements in such a system may be iAPX 286 subsystems. Chapter Two outlines how the iAPX 286 system architecture supports a system bus, and details some of the considerations for interfacing an iAPX 286 system to a multiprocessor system bus.

This chapter expands on the description given in Chapter Two and describes how to interface an iAPX 286 subsystem to a multimaster system bus, using the IEEE 796 Multibus protocols.

- The first section of this chapter discusses some of the reasons for using a system bus, and describes the tradeoffs between placing particular system resources on a local bus or the system bus.
- The second section describes a particular implementation of a multiprocessor system bus, the IEEE 796 (Intel Multibus) system bus. This section also details how specific Intel components can make the design and implementation of a Multibus interface easy and efficient.
- The third section describes three of the four principal considerations that must be taken into account when designing an iAPX 286 interface to the Multibus. This section describes:
 - A. The decoding of memory and I/O references onto either the iAPX 286 local bus or the Multibus. A technique for mapping Multibus I/O into the iAPX 286 memory space is explained, and a byte-swapping circuit is introduced to comply with the Multibus requirements for byte transfers.
 - B. The decoding of interrupts and interrupt acknowledge sequences onto either the local bus or the Multibus.
- The fourth section describes the use of the 82289 Bus Arbiter in implementing a Multibus interface. The Bus Arbiter coordinates the contention of the iAPX 286 system for the Multibus, and controls the release of the Multibus to other Multibus processors following iAPX 286 usage.
- The fifth section contains a timing analysis of the Multibus interface, and summarizes the key Multibus parameters, describing how the iAPX 286 system must be configured to meet these requirements.
- The sixth section discusses using dual-port memories with the Multibus system bus interface, and describes the handling of LOCK signals.

THE SYSTEM-BUS CONCEPT

Previous chapters considered single-bus systems in which a single iAPX 286 processor connects to memory, I/O, and processor extensions. This chapter introduces the system bus concept, which allows several single-bus systems to be connected into a more-powerful multiprocessing system.

In a single-bus system, a local bus connects processing elements with memory and I/O subsystems where each processing element can access any resource on the local bus. However, since only one processing element at a time can use the local bus, system throughput cannot be improved by adding more processing elements. For this reason, a local bus typically contains only one general-purpose processing element and perhaps one or more dedicated processors, along with memory and other resources required by the processors.

A system bus can connect several processing subsystems, each of which may have their own local bus and private resources. The system bus may also connect system resources such as memory and I/O, which are shared equally between processing subsystems. In this way, the system bus supports multiprocessing. Since each of the processing subsystems can perform simultaneous data transfers on their respective local buses, total system throughput can be increased greatly over that of a single-bus system.

The system bus also establishes a standard interface, allowing computer systems to be expanded modularly using components from different vendors. The Intel Multibus, for example, has over 100 vendors supplying over 800 board-level products that are compatible with the Multibus interface. Using the standard Multibus protocols, a wide variety of I/O devices and memory subsystems are available to expand the capabilities of iAPX 286 systems.

Although the bulk of this chapter describes the Multibus system bus, the concepts discussed here are applicable to any system bus.

The Division of Resources

The heart of the system-bus concept is the division of resources between a processing element's local bus and the system bus. Typically, an iAPX 286 subsystem that interfaces to a system bus will have some amount of memory and perhaps other resources connected to its local bus. An important point to consider in designing a multiprocessing system is how resources will be divided between local buses and the system bus. An iAPX 286 subsystem must communicate with resources on both its local bus and the system bus. There are a number of tradeoffs between placing a particular system resource such as memory on a single processor's local bus, or placing the resource on the system bus.

LOCAL RESOURCES

Resources on a local bus are accessible only by the processor controlling that local bus. This may increase reliability, for such resources are isolated from the effects of failures occurring in other parts of a system. System throughput can also be increased by using local resources, since the local processor does not have to contend with other processors for the use of these resources. In systems where several processors each have their own local memory, multiple tasks can execute in parallel because each processor is fetching instructions from a separate path. In an iAPX 286 system, local memory allows the 80286 to use its capability for overlapped memory cycles to the best effect: 80286 bus cycles can be performed in the least possible time.

Of course, the cost of implementing a separate memory subsystem for each processor must be considered. In addition, some memory will have to be connected to the system bus, to allow different processing elements to communicate with each other. Occasionally-used processor resources may be more cost-effective if they are connected to the system bus, where their use can be shared among several processors.

SYSTEM RESOURCES

Resources directly connected to the system bus are accessible to all processing elements on the system bus; therefore, they can be shared efficiently among different processors. Memory resources connected to the system bus allow processors to pass blocks of data between each other efficiently and thus communicate asynchronously.

A disadvantage of placing resources on the system bus is that a single processor may have to contend with other processors for access to the bus, resulting in slower access times and reduced system

throughput. Using system memory also involves a risk of memory corruption because it is conceivable that one processing element may overwrite data being used by another.

In view of these tradeoffs, designers must carefully consider which resources (or how much memory) they will place on a subsystem's local bus, and which or how much to place on the system bus. These choices affect system reliability, integrity, throughput and performance, and often depend on the requirements of the particular target system.

THE IEEE 796 MULTIBUS®—A MULTIMASTER SYSTEM BUS

The Intel Multibus (IEEE 796 Standard) is an example of a proven, industry-standard multiprocessing system bus that is well-tailored for iAPX 286 systems. A wide variety of Multibus-compatible I/O subsystems, memory boards, general-purpose processing boards, and dedicated-function boards are available from Intel to speed product development while ensuring bus-level compatibility. Designers who choose the Multibus protocols in their system bus have a ready supply of system components available for use in their products.

The Multibus protocols are completely described in the Intel Multibus Specification, Order Number 9800683-04.

The job of interfacing an iAPX 286 subsystem to the Multibus is made relatively simple by using several components specifically adapted to handling the Multibus protocols. These interface components include:

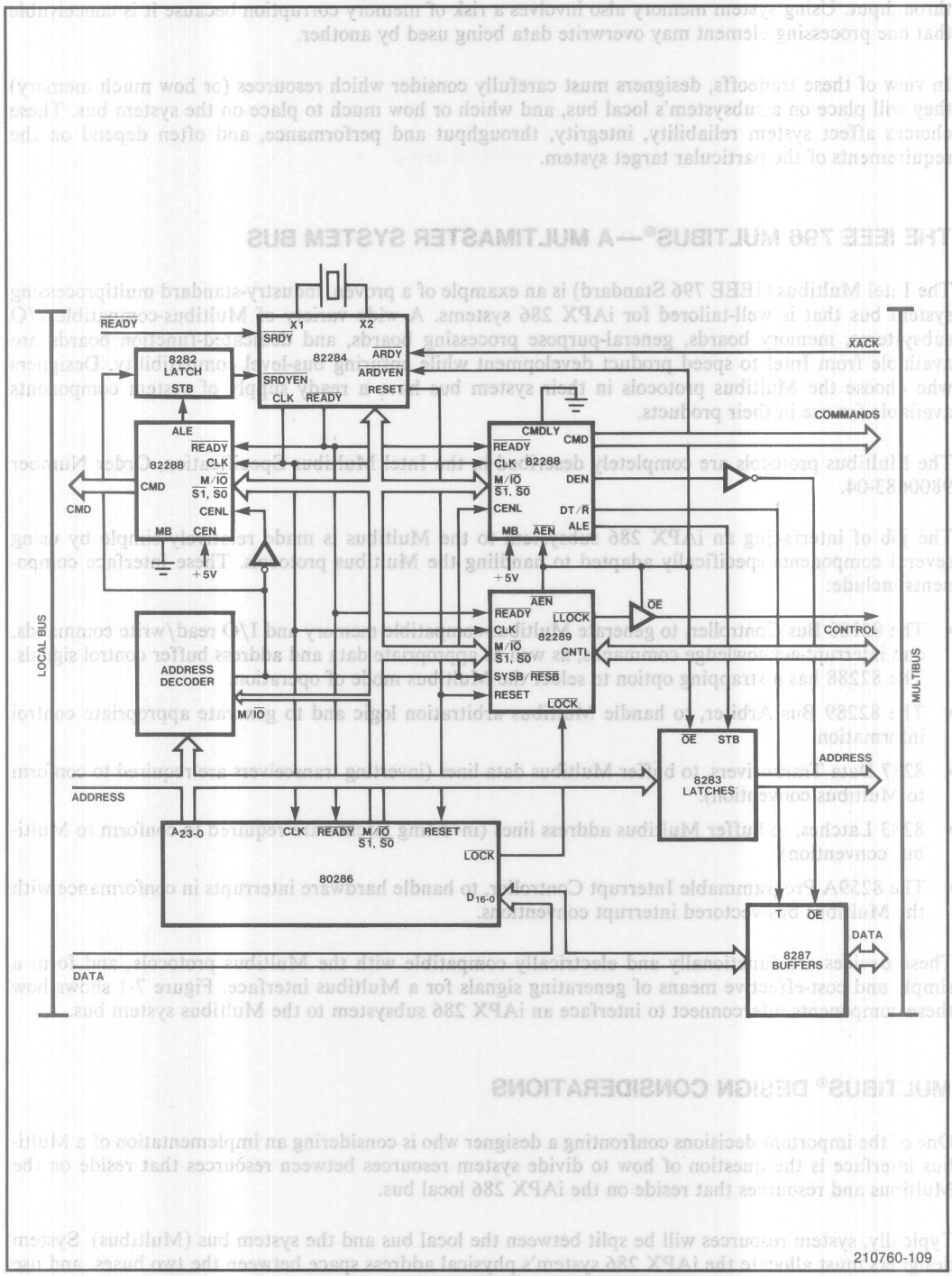
- The 82288 Bus Controller, to generate Multibus-compatible memory and I/O read/write commands, and interrupt-acknowledge commands, as well as appropriate data and address buffer control signals. The 82288 has a strapping option to select the Multibus mode of operation.
- The 82289 Bus Arbiter, to handle Multibus arbitration logic and to generate appropriate control information.
- 8287 Data Transceivers, to buffer Multibus data lines (inverting transceivers are required to conform to Multibus convention).
- 8283 Latches, to buffer Multibus address lines (inverting latches are required to conform to Multibus convention).
- The 8259A Programmable Interrupt Controller, to handle hardware interrupts in conformance with the Multibus bus-vectored interrupt conventions.

These devices are functionally and electrically compatible with the Multibus protocols, and form a simple and cost-effective means of generating signals for a Multibus interface. Figure 7-1 shows how these components interconnect to interface an iAPX 286 subsystem to the Multibus system bus.

MULTIBUS® DESIGN CONSIDERATIONS

One of the important decisions confronting a designer who is considering an implementation of a Multibus interface is the question of how to divide system resources between resources that reside on the Multibus and resources that reside on the iAPX 286 local bus.

Typically, system resources will be split between the local bus and the system bus (Multibus). System designers must allocate the iAPX 286 system's physical address space between the two buses, and use this information to select either the iAPX 286 local bus or the system bus for each bus cycle.



210760-109

Figure 7-1. Local Bus and MULTIBUS® Interface for the iAPX 286

Three different types of bus operations must be considered when designing this decoding function:

1. Memory operations
2. I/O operations
3. Interrupt-acknowledge sequences

For memory and I/O operations, several additional features of the Multibus must be considered. The following paragraphs describe each of these types in detail.

Memory Operations

The decoding of memory operations to select either the local bus or the system bus is relatively straightforward. Typically, memory resources that reside on the system bus can be allocated to particular address windows, or ranges. An address decoder can be used to decode the ranges of addresses for memory that resides on the system bus, and the output of this decoder then selects either the system bus or the local bus for the current bus cycle.

Figure 7-3 illustrates how the output of the address decoder drives the 82288 Bus Controller and 82289 Bus Arbiter circuits to selectively activate either the local bus or the system bus. Since both the CENL input of the Bus Controller and the SYSB/$\overline{\text{RESB}}$ input of the Bus Arbiter are internally-latched, no additional latches are required to maintain selection of the bus for the remainder of the current bus cycle.

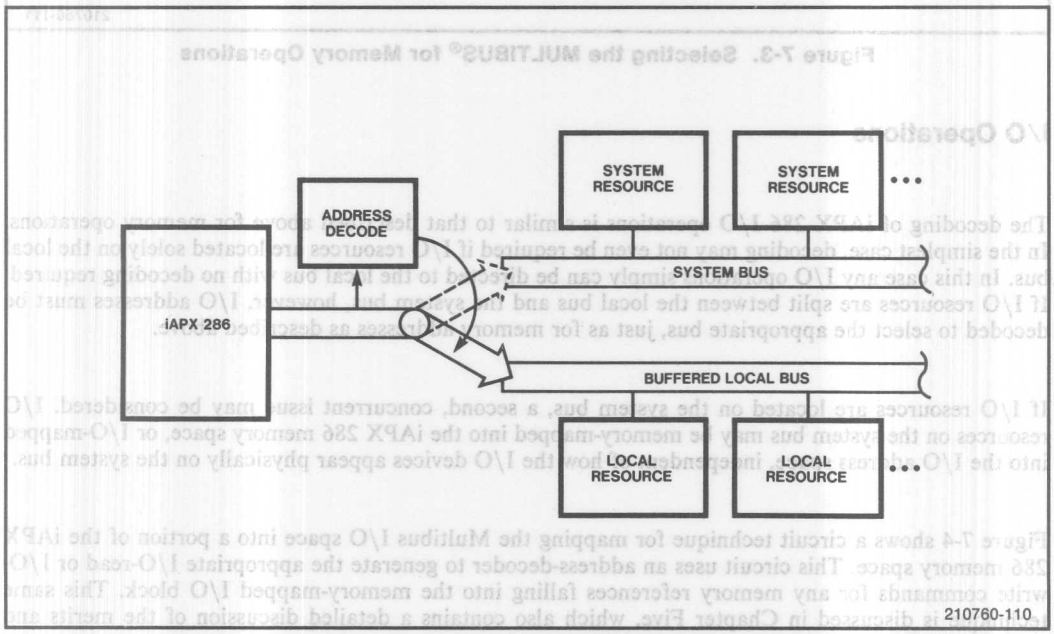


Figure 7-2. Decoders Select the Local vs. the System Bus

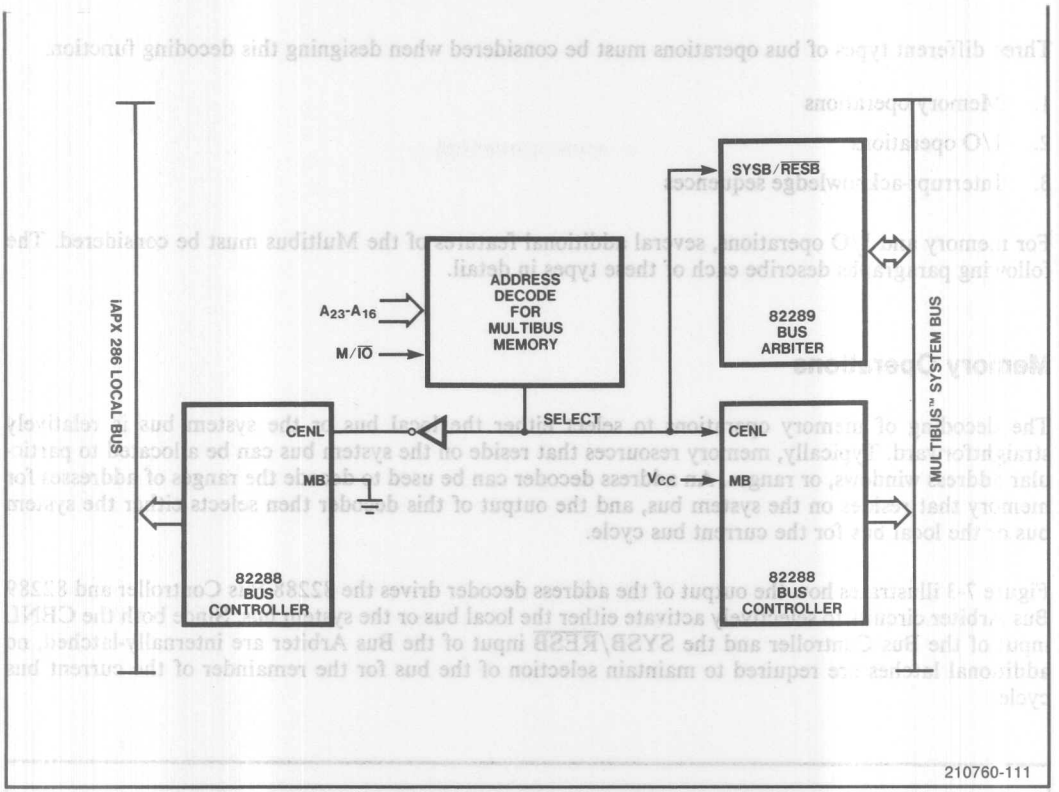


Figure 7-3. Selecting the MULTIBUS® for Memory Operations

I/O Operations

The decoding of iAPX 286 I/O operations is similar to that described above for memory operations. In the simplest case, decoding may not even be required if I/O resources are located solely on the local bus. In this case any I/O operations simply can be directed to the local bus with no decoding required. If I/O resources are split between the local bus and the system bus, however, I/O addresses must be decoded to select the appropriate bus, just as for memory addresses as described above.

If I/O resources are located on the system bus, a second, concurrent issue may be considered. I/O resources on the system bus may be memory-mapped into the iAPX 286 memory space, or I/O-mapped into the I/O address space, independent of how the I/O devices appear physically on the system bus.

Figure 7-4 shows a circuit technique for mapping the Multibus I/O space into a portion of the iAPX 286 memory space. This circuit uses an address-decoder to generate the appropriate I/O-read or I/O-write commands for any memory references falling into the memory-mapped I/O block. This same technique is discussed in Chapter Five, which also contains a detailed discussion of the merits and tradeoffs between memory-mapped and I/O-mapped devices.

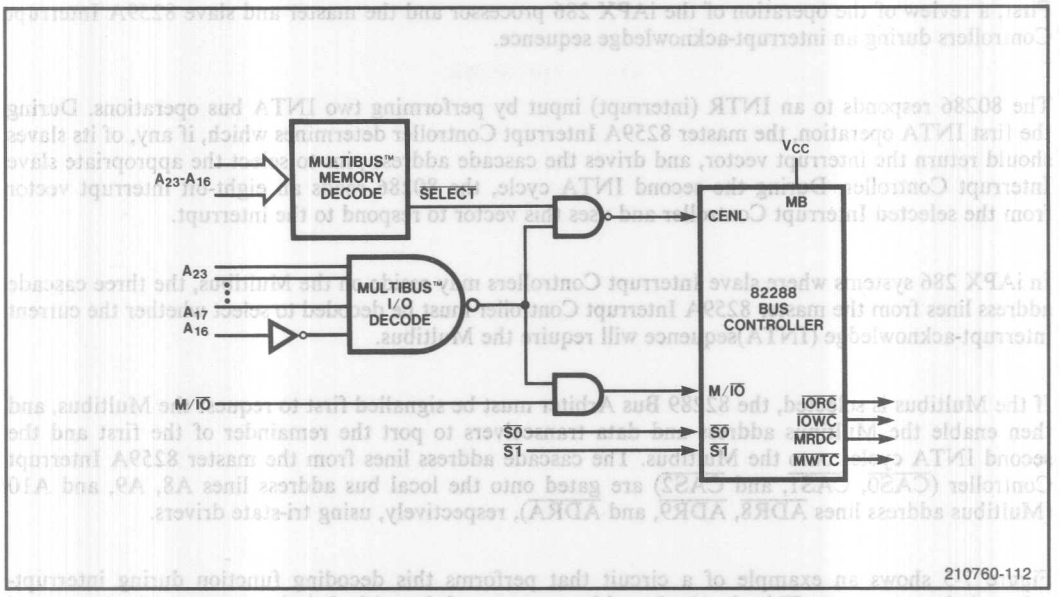


Figure 7-4. Memory-Mapping the MULTIBUS® I/O

Interrupt-Acknowledge to Cascaded Interrupt Controllers

A third issue related to the division of system resources between the system bus (Multibus) and the local bus is that of mapping interrupts between the Multibus and the local data bus. (You may recall that when an interrupt is received by the 80286, the iAPX 286 interrupt-acknowledge sequence uses the data bus to fetch an 8-bit interrupt vector from the interrupting 8259A Programmable Interrupt Controller.)

Designers may encounter three possible configurations:

1. All of the subsystem's Interrupt Controllers (one master and perhaps one or more slaves) reside on the local bus, and, therefore, all interrupt-acknowledge cycles are routed to the local bus.
2. All "terminal" Interrupt Controllers reside on the Multibus system bus (either all 8259A Interrupt Controllers reside on the Multibus bus, or a master 8259A resides on the local bus but services interrupts *only* from slave 8259A Interrupt Controllers that themselves reside on the Multibus), and so all interrupt-acknowledge cycles are automatically routed to the Multibus.
3. Some slave Interrupt Controllers reside on the subsystem's local bus, and others reside on the Multibus. In this case, the subsystem's hardware must decode the Master Interrupt Controller's cascade address and select the appropriate bus for the interrupt-acknowledge cycles.

In the first two cases, the implementation is relatively straightforward; all interrupt-acknowledge bus operations can be automatically directed onto the appropriate bus. If one of these two configurations is envisioned for the system you are designing, you can skip this section and continue on. If, however, your iAPX 286 subsystem will have a master 8259A Interrupt Controller on the local bus and *at least one* slave 8259A Interrupt Controller connected to the Multibus, you should read this section to discover how to handle the direction of interrupt-acknowledge bus cycles onto either the Multibus or the local bus.

First, a review of the operation of the iAPX 286 processor and the master and slave 8259A Interrupt Controllers during an interrupt-acknowledge sequence.

The 80286 responds to an INTR (interrupt) input by performing two INTA bus operations. During the first INTA operation, the master 8259A Interrupt Controller determines which, if any, of its slaves should return the interrupt vector, and drives the cascade address pins to select the appropriate slave Interrupt Controller. During the second INTA cycle, the 80286 reads an eight-bit interrupt vector from the selected Interrupt Controller and uses this vector to respond to the interrupt.

In iAPX 286 systems where slave Interrupt Controllers may reside on the Multibus, the three cascade address lines from the master 8259A Interrupt Controller must be decoded to select whether the current interrupt-acknowledge (INTA)sequence will require the Multibus.

If the Multibus is selected, the 82289 Bus Arbiter must be signalled first to request the Multibus, and then enable the Multibus address and data transceivers to port the remainder of the first and the second INTA cycles onto the Multibus. The cascade address lines from the master 8259A Interrupt Controller ($\overline{CAS0}$, $\overline{CAS1}$, and $\overline{CAS2}$) are gated onto the local bus address lines A8, A9, and A10 (Multibus address lines $\overline{ADR8}$, $\overline{ADR9}$, and $\overline{ADRA}$), respectively, using tri-state drivers.

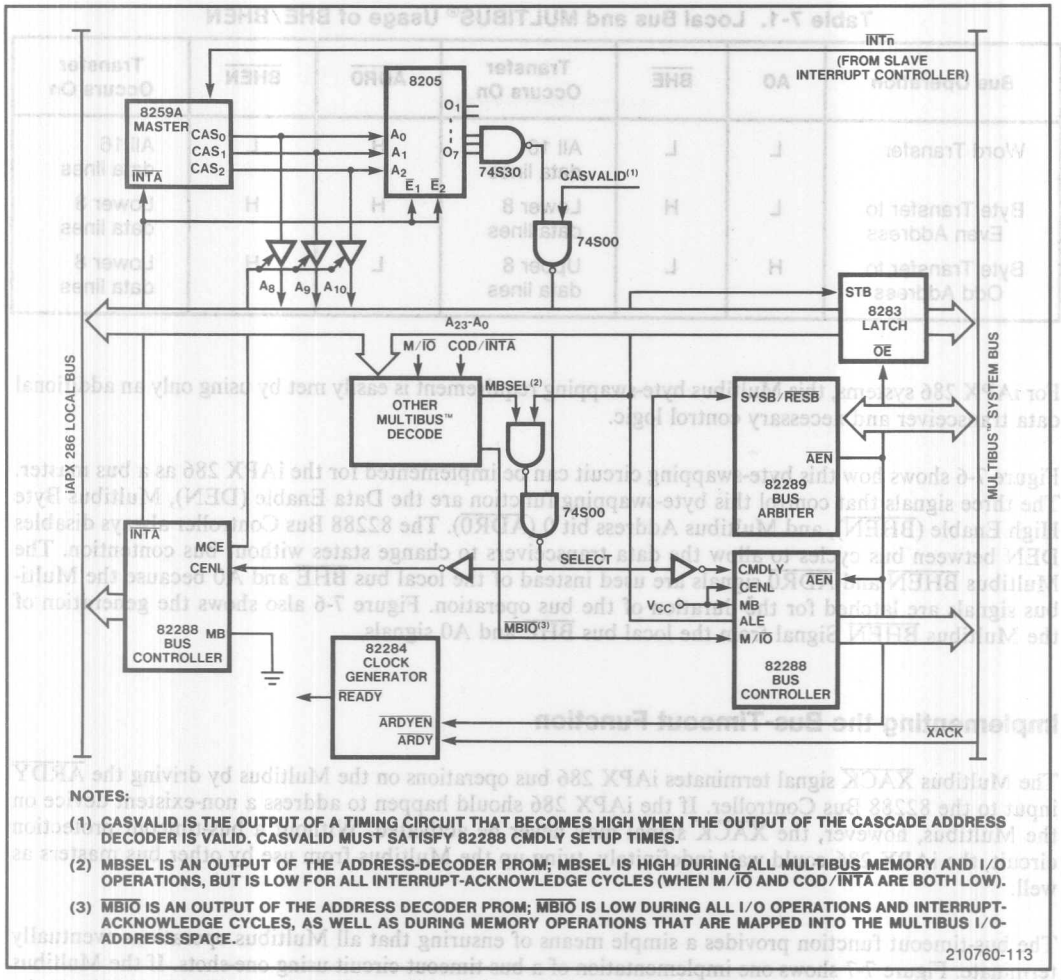
Figure 7-5 shows an example of a circuit that performs this decoding function during interrupt-acknowledge sequences. This circuit also addresses some of the critical timing necessary in the case of a master 8259A Interrupt Controller gating a cascade address onto the Multibus.

The timing of the basic interrupt-acknowledge cycle is described in Chapter Three. During the first INTA cycle, the cascade address from the master 8259A Interrupt Controller residing on the local bus becomes valid within a maximum of 565 ns following the assertion of the $\overline{INTA}$ signal from the Bus Controller. Once this address becomes valid, it must be decoded to determine whether the remainder of the INTA sequence should remain on the local bus, or should take place on the Multibus. Using the cascade-decode logic shown in Figure 7-5, the decode delay adds an additional 30 ns, for a total of 595 ns from $\overline{INTA}$ to bus-select valid. The local bus Ready logic must therefore insert at least 4 wait states into the first INTA cycle before the cascade address becomes valid and the local- or Multibus-select becomes valid.

If the local bus is selected for the remainder of the INTA sequence, the first INTA cycle can be terminated immediately. If the Multibus is selected, however, the first INTA bus cycle must still be extended while it is ported onto the Multibus in order to properly condition the slave Interrupt Controllers on the Multibus.

The circuit shown in Figure 7-5 shows how the Multibus 82289 Bus Arbiter and 82288 Bus Controller are selected to allow this "delayed startup" of the Multibus INTA bus cycle. The CMDLY input to the 82288 is used as a select input that is sampled repetitively by the 82288; the CENL input can be tied high. Once the Multibus has been selected, the Multibus $\overline{XACK}$ signal should be used to terminate the bus operation.

During the second INTA bus cycle, the Master Cascade Enable (MCE) output of the 82288 Bus Controller becomes active to gate the cascade address onto address lines A8, A9, and A10. This MCE enable signal stays active one clock cycle longer than the ALE signal, allowing the cascade address to be properly captured in the Multibus address latches. The MCE signal becomes active only during INTA cycles—no additional logic is required to properly enable the cascade address onto the address bus during INTA cycles.



NOTES:

(1) CASVALID IS THE OUTPUT OF A TIMING CIRCUIT THAT BECOMES HIGH WHEN THE OUTPUT OF THE CASCADE ADDRESS DECODER IS VALID. CASVALID MUST SATISFY 82288 CMDLY SETUP TIMES.

(2) MBSEL IS AN OUTPUT OF THE ADDRESS-DECODER PROM; MBSEL IS HIGH DURING ALL MULTIBUS MEMORY AND I/O OPERATIONS, BUT IS LOW FOR ALL INTERRUPT-ACKNOWLEDGE CYCLES (WHEN M/$\overline{IO}$ AND COD/$\overline{INTA}$ ARE BOTH LOW).

(3) $\overline{MBIO}$ IS AN OUTPUT OF THE ADDRESS DECODER PROM; $\overline{MBIO}$ IS LOW DURING ALL I/O OPERATIONS AND INTERRUPT-ACKNOWLEDGE CYCLES, AS WELL AS DURING MEMORY OPERATIONS THAT ARE MAPPED INTO THE MULTIBUS I/O-ADDRESS SPACE.

210760-113

Figure 7-5. Decoding Interrupt-Acknowledge Sequences

Byte-Swapping During MULTIBUS® Byte Transfers

The Multibus standard specifies that during all byte transfers, the data must be transferred on the lower-eight data lines (Multibus $\overline{DAT0}$ through $\overline{DAT7}$), whether the data is associated with an even (low-byte) or odd (high-byte) address. For 16-bit systems, this requirement means that during byte transfers to odd addresses, the data byte must be "swapped" from the high data lines (local bus D8 through D15) onto the low data lines (D0 through D7) before being placed on the Multibus, and then "swapped" back onto the high data lines when being read from the Multibus. This byte-swapping requirement maintains compatibility between 8-bit and 16-bit systems sharing the same Multibus.

Because of this byte-swapping requirement, the Multibus $\overline{BHEN}$ signal differs in definition from the $\overline{BHE}$ signal on the iAPX 286 local bus. Table 7-1 illustrates the differences between these two signals. Notice the difference that appears for byte transfers to odd addresses.

Table 7-1. Local Bus and MULTIBUS® Usage of BHE/BHEN

Bus Operation	A0	$\overline{BHE}$	Transfer Occurs On	$\overline{ADR0}$	$\overline{BHEN}$	Transfer Occurs On
Word Transfer	L	L	All 16 data lines	H	L	All 16 data lines
Byte Transfer to Even Address	L	H	Lower 8 data lines	H	H	Lower 8 data lines
Byte Transfer to Odd Address	H	L	Upper 8 data lines	L	H	Lower 8 data lines

For iAPX 286 systems, this Multibus byte-swapping requirement is easily met by using only an additional data transceiver and necessary control logic.

Figure 7-6 shows how this byte-swapping circuit can be implemented for the iAPX 286 as a bus master. The three signals that control this byte-swapping function are the Data Enable (DEN), Multibus Byte High Enable ($\overline{BHEN}$), and Multibus Address bit 0 ($\overline{ADR0}$). The 82288 Bus Controller always disables DEN between bus cycles to allow the data transceivers to change states without bus contention. The Multibus $\overline{BHEN}$ and $\overline{ADR0}$ signals are used instead of the local bus $\overline{BHE}$ and A0 because the Multibus signals are latched for the duration of the bus operation. Figure 7-6 also shows the generation of the Multibus $\overline{BHEN}$ Signal from the local bus $\overline{BHE}$ and A0 signals.

Implementing the Bus-Timeout Function

The Multibus $\overline{XACK}$ signal terminates iAPX 286 bus operations on the Multibus by driving the $\overline{ARDY}$ input to the 82288 Bus Controller. If the iAPX 286 should happen to address a non-existent device on the Multibus, however, the $\overline{XACK}$ signal may never be activated. Without a bus-timeout protection circuit, the iAPX 286 could wait indefinitely, tying up the Multibus from use by other bus masters as well.

The bus-timeout function provides a simple means of ensuring that all Multibus operations eventually terminate. Figure 7-7 shows one implementation of a bus-timeout circuit using one-shots. If the Multibus $\overline{XACK}$ signal is not returned within a finite period, the bus-timeout signal is activated to terminate the bus operation.

When a bus-timeout occurs, the data read by the 80286 may not be valid. For this reason, the bus-timeout circuit may also be used to generate an interrupt to prevent software from using invalid data.

Power Failure Considerations

The Multibus interface provides a means of handling power failures by defining a Power Fail Interrupt ($\overline{PFIN}$) signal and other status lines, and by making provisions for secondary or backup power supplies.

Typically, the Power Fail Interrupt ($\overline{PFIN}$) from the Multibus is connected to the NMI interrupt request line of the CPU. When a power failure is about to occur, this interrupt enables the 80286 CPU to immediately save its environment before falling voltages and the Multibus Memory Protect ($\overline{MPRO}$) signal prevent any further memory activity. In systems with memory backup power or non-volatile memory, the 80286 environment can be saved for the duration of the powerfail condition.

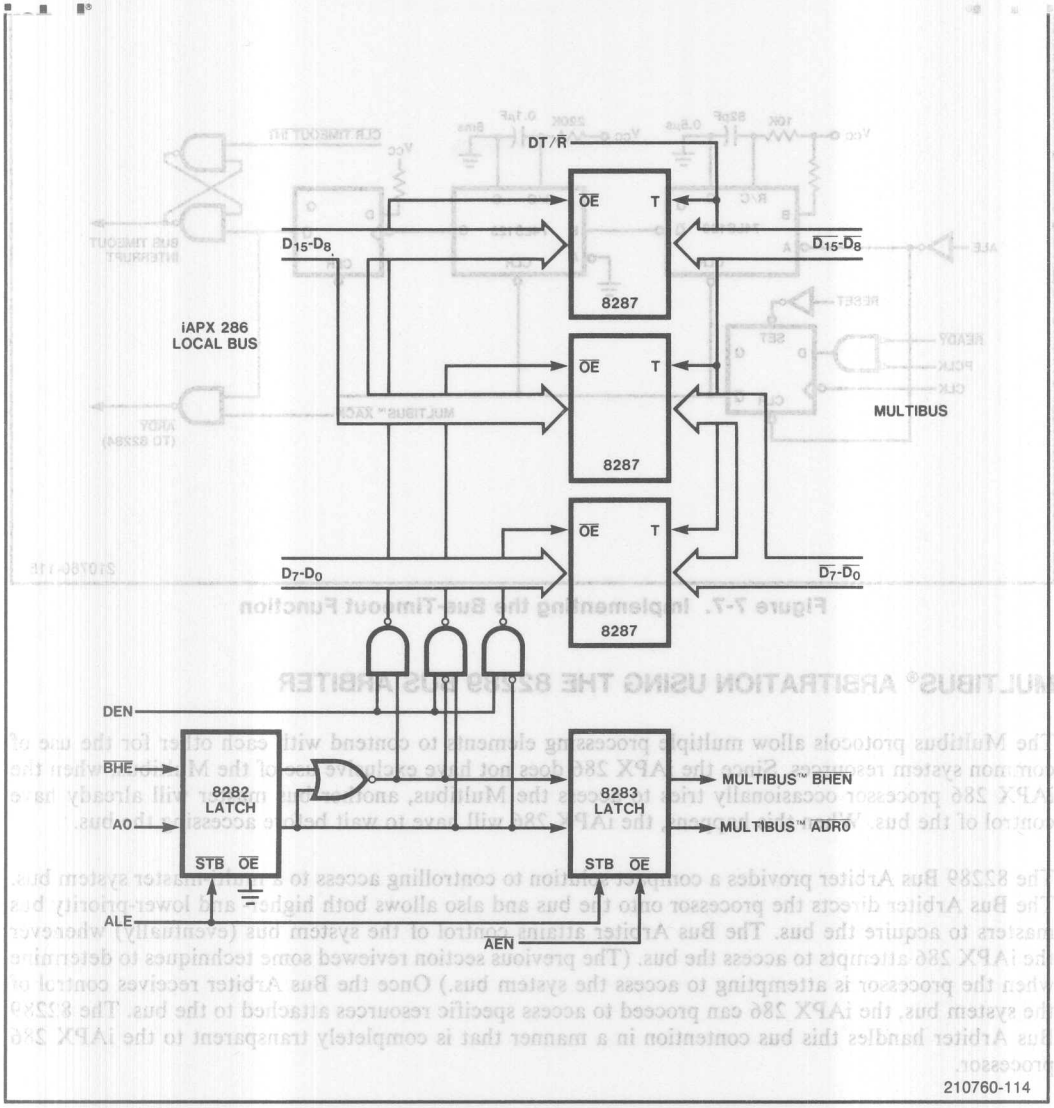


Figure 7-6. Byte-Swapping at the MULTIBUS® Interface

When the AC power is restored, the power-up RESET sequence of the 80286 CPU can check the status of the Multibus Power Fail Sense Latch ($\overline{PFSN}$) to see if a previous power failure has occurred. If this latch is set low, the 80286 can branch to a powerup routine that resets the latch using Power Fail Sense Reset ($\overline{PFSR}$), restores its environment, and resumes execution.

Further guidelines for designing iAPX 286 systems with power-failure features are contained in the Intel Multibus Specification referred to previously.

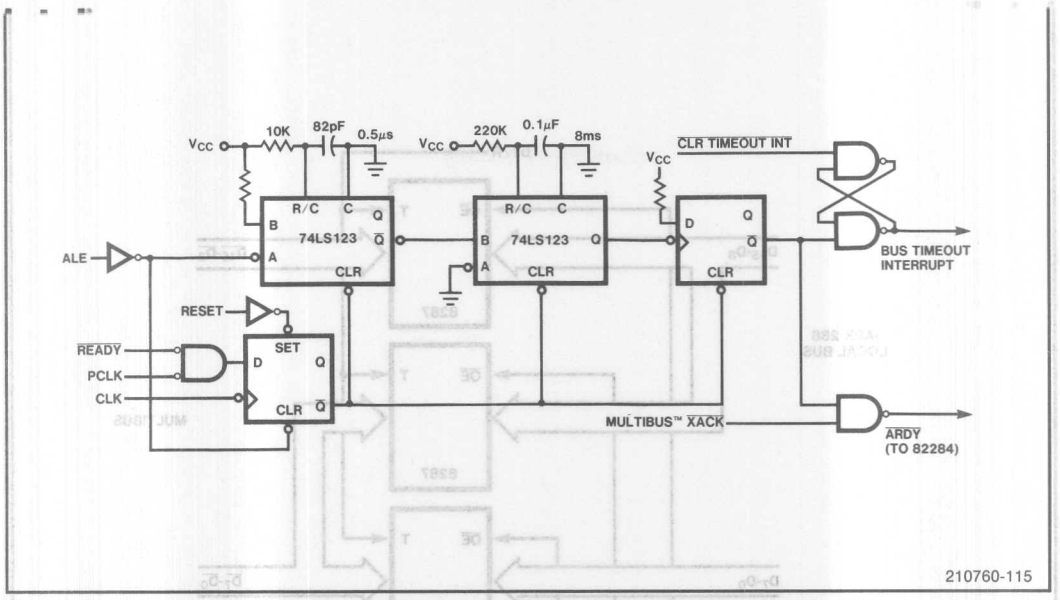


Figure 7-7. Implementing the Bus-Timeout Function

MULTIBUS® ARBITRATION USING THE 82289 BUS ARBITER

The Multibus protocols allow multiple processing elements to contend with each other for the use of common system resources. Since the iAPX 286 does not have exclusive use of the Multibus, when the iAPX 286 processor occasionally tries to access the Multibus, another bus master will already have control of the bus. When this happens, the iAPX 286 will have to wait before accessing the bus.

The 82289 Bus Arbiter provides a compact solution to controlling access to a multi-master system bus. The Bus Arbiter directs the processor onto the bus and also allows both higher- and lower-priority bus masters to acquire the bus. The Bus Arbiter attains control of the system bus (eventually) whenever the iAPX 286 attempts to access the bus. (The previous section reviewed some techniques to determine when the processor is attempting to access the system bus.) Once the Bus Arbiter receives control of the system bus, the iAPX 286 can proceed to access specific resources attached to the bus. The 82289 Bus Arbiter handles this bus contention in a manner that is completely transparent to the iAPX 286 processor.

Gaining Control of the MULTIBUS®

In an iAPX 286 subsystem using an 82289 Bus Arbiter, the iAPX 286 processor issues commands as though it had exclusive use of the system bus. The Bus Arbiter keeps track of whether the subsystem indeed has control of the system bus, and if not, prevents the 82288 Bus Controller and address latches from accessing the bus. The Bus Arbiter also inhibits the 82284 Clock Generator, forcing the 82286 processor into one or more wait states.

When the Bus Arbiter receives control of the system bus, the Bus Arbiter enables the $\overline{\text{ARDY}}$ input of the Clock Generator, and enables the 82288 Bus Controller and address latches to drive the system

bus. Once the system bus transfer is complete, a transfer-acknowledge ($\overline{XACK}$) signal is returned by the Multibus, signalling to the processor that the transfer cycle has completed. In this manner, the Bus Arbiter multiplexes one bus master onto the multi-master bus and avoids contention between bus masters.

Since many bus masters can be on a multi-master system bus, some means must be provided for resolving priority between bus masters simultaneously requesting the bus. Figure 7-8 shows two common priority-resolution schemes: a serial-priority and a parallel-priority technique.

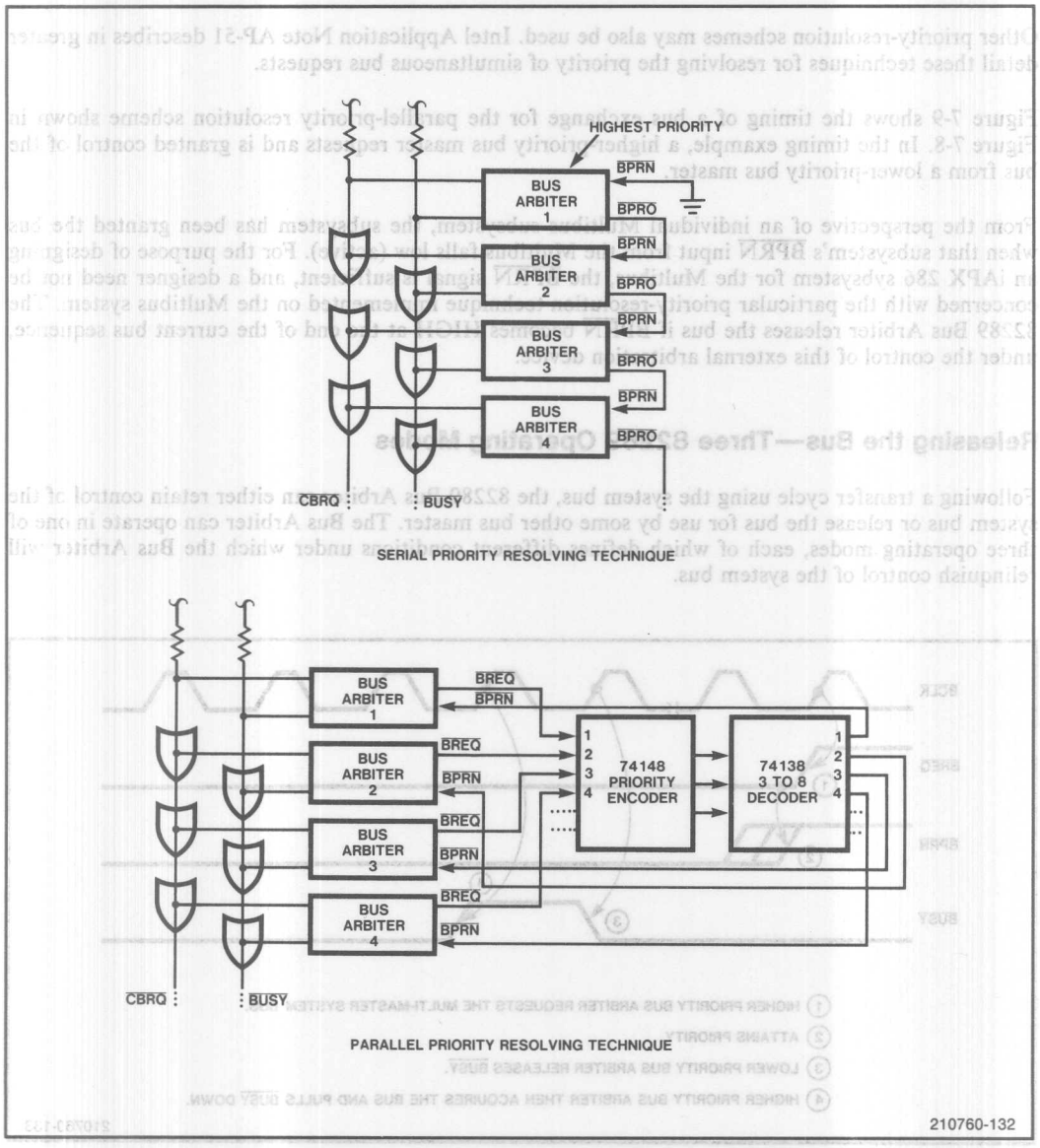


Figure 7-8. Two Bus Priority-Resolution Techniques

The serial-priority resolution technique consists of a daisy-chain of the 82289 Bus Priority In ($\overline{\text{BPRN}}$) and Bus Priority Out ($\overline{\text{BPRO}}$) signals. Due to delays in the daisy-chain, however, only a limited number of bus masters can be accommodated using this technique.

In the parallel-priority resolution technique, each 82289 Bus Arbiter makes independent bus requests using its Bus Request ($\overline{\text{BREQ}}$) signal line. An external bus-priority resolution circuit determines the highest-priority bus master requesting the bus, and grants that master control of the bus by setting $\overline{\text{BPRN}}$ low to that bus master. Any number of bus masters can be accommodated using this technique, limited only by the complexity of the external resolution circuitry.

Other priority-resolution schemes may also be used. Intel Application Note AP-51 describes in greater detail these techniques for resolving the priority of simultaneous bus requests.

Figure 7-9 shows the timing of a bus exchange for the parallel-priority resolution scheme shown in Figure 7-8. In the timing example, a higher-priority bus master requests and is granted control of the bus from a lower-priority bus master.

From the perspective of an individual Multibus subsystem, the subsystem has been granted the bus when that subsystem's $\overline{\text{BPRN}}$ input from the Multibus falls low (active). For the purpose of designing an iAPX 286 sybsystem for the Multibus, the $\overline{\text{BPRN}}$ signal is sufficient, and a designer need not be concerned with the particular priority-resolution technique implemented on the Multibus system. The 82289 Bus Arbiter releases the bus if $\overline{\text{BPRN}}$ becomes HIGH at the end of the current bus sequence, under the control of this external arbitration device.

Releasing the Bus—Three 82289 Operating Modes

Following a transfer cycle using the system bus, the 82289 Bus Arbiter can either retain control of the system bus or release the bus for use by some other bus master. The Bus Arbiter can operate in one of three operating modes, each of which defines different conditions under which the Bus Arbiter will relinquish control of the system bus.

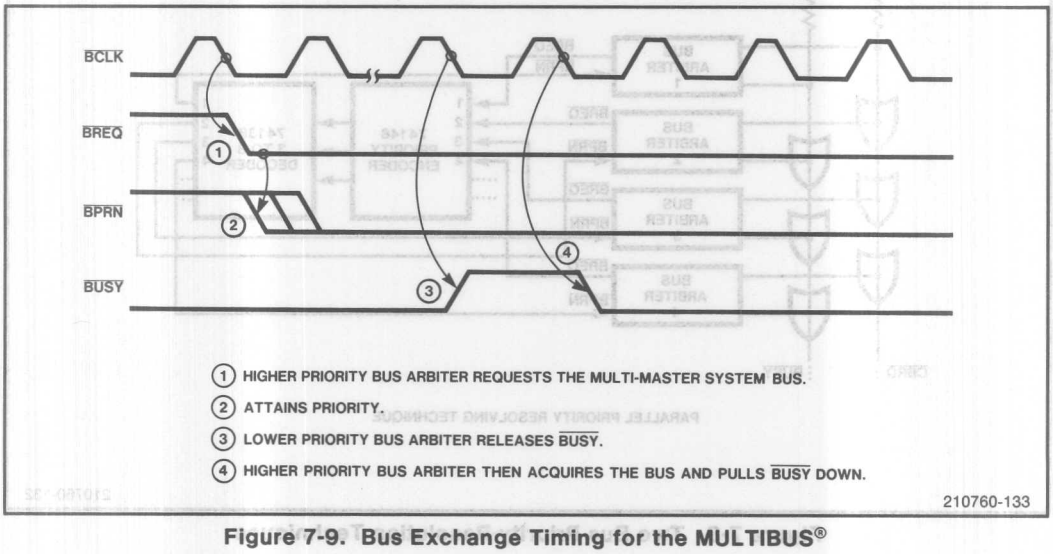


Figure 7-9. Bus Exchange Timing for the MULTIBUS®

Table 7-2 defines the three operating modes of the 82289 Bus Arbiter, and describes the conditions under which the Bus Arbiter relinquishes control of the Multibus. The following sections describe how to configure the Bus Arbiter in any one of these modes, and also describe a way to switch the Bus Arbiter between Modes 2 and 3 under program control from the iAPX 286.

The decision to configure the 82289 Bus Arbiter in one of these three modes, or to configure the Bus Arbiter in a fourth manner, allowing switching between modes 2 and 3, is a choice left up to the individual designer. This choice may affect the throughput of the individual subsystem, as well as the throughput of other subsystems sharing the bus, and system throughput as a whole.

This performance impact occurs because a multi-processor system may have appreciable overhead when a processor requests and takes control of the Multibus. Figure 7-10 illustrates the effect of bus set-up and hold times on bus efficiency and throughput.

When to Use the Different Modes

The various operating modes of the Bus Arbiter allow the designer to optimize a subsystem's use of the Multibus to its own needs and to the needs of the system as a whole.

Mode 1, for example, would be adequate for a sybsystem that needed to access the Multibus only occasionally; by releasing the bus after each transfer cycle, the subsystem would minimize its impact on the throughput of other subsystems using the bus.

Mode 2 would be ideal for a subsystem that shared the Multibus with a pool of other bus masters, all of roughly equal priority, and all equally likely to request the bus at any given time. The performance improvement of retaining the bus in case of a second or subsequent access to the bus would be matched to the performance impact of other bus masters, who may have to request the bus from the controlling Bus Arbiter, and thus suffer the ensuing hold delays.

Mode 3 would be ideal for a system that is likely to be using the Multibus a high percentage of the time. The performance advantages of retaining control of the bus would outweigh the chances of other, lower-priority devices waiting longer periods to gain access to the Multibus.

Table 7-2. Three 82289 Operating Modes for Releasing the Bus

Operating Mode	Conditions under which the Bus Arbiter releases the system bus*
Mode 1	The Bus Arbiter releases the bus at the end of each transfer cycle.
Mode 2	The Bus Arbiter retains the bus until: • a higher-priority bus master requests the bus, driving $\overline{\text{BPRN}}$ high. • a lower-priority bus master requests the bus by pulling the $\overline{\text{CBRQ}}$ low.
Mode 3	The Bus Arbiter retains the bus until a higher-priority bus master requests the bus, driving $\overline{\text{BPRN}}$ high. (In this mode, the Bus Arbiter ignores the $\overline{\text{CBRQ}}$ input.

*The $\overline{\text{LOCK}}$ input to the Bus Arbiter can be used to override any of the conditions shown in the table. While $\overline{\text{LOCK}}$ is asserted, the Bus Arbiter will retain control of the system bus.

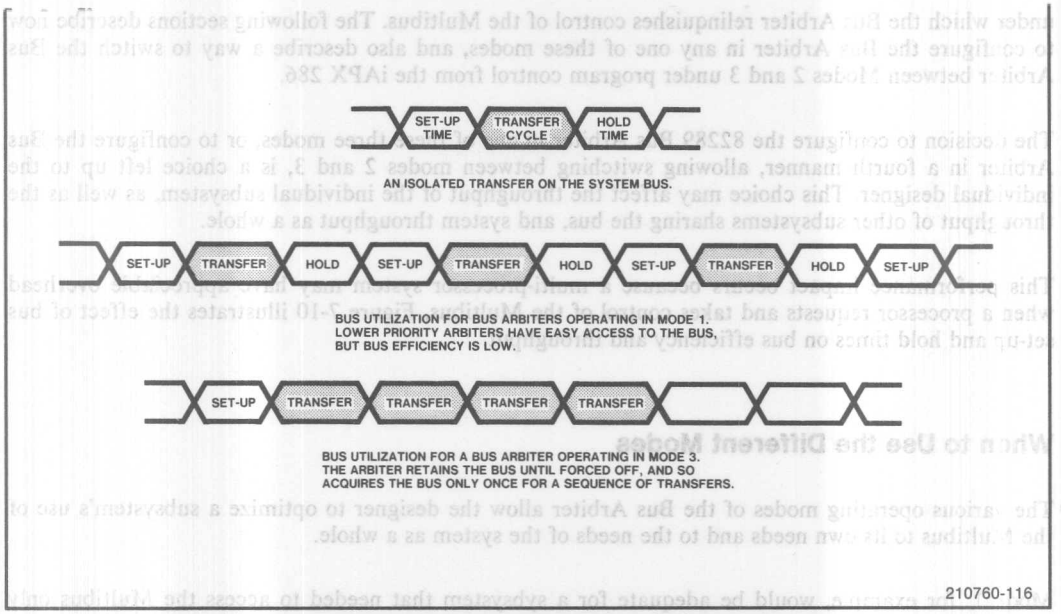


Figure 7-10. Effects of Bus Contention on Bus Efficiency

A fourth alternative, that of allowing a processor to switch its Bus Arbiter between Mode 2 and Mode 3, offers even more flexibility in optimizing system performance where such performance might be subject to experimentation, or where Multibus traffic might be sporadic.

To summarize the preceeding discussion, the decision to use one or another of the four Bus Arbiter configurations depends on the relative priorities of processors sharing the bus, and on the anticipated traffic each processor may have for the bus. In order to optimize the sharing of the Multibus between several bus masters, designers are urged to experiment with the different modes of each Bus Arbiter in the system.

Configuring the 82289 Operating Modes

A designer can configure the Bus Arbiter in four ways:

1. The Bus Arbiter can be configured to operate in Mode 1, releasing the bus at the end of each transfer cycle.
2. The Bus Arbiter can be configured in Mode 2, retaining the bus until either a higher- or lower-priority bus master requests the bus.
3. The Bus Arbiter can be configured in Mode 3, retaining the bus until a higher-priority bus master requests the bus.
4. The Bus Arbiter can be configured so that it can be switched between Mode 2 and Mode 3 under software control of the iAPX 286. This last configuration is more complex than the first three, requiring that a parallel port or addressable latch be used to drive one of the strapping pins of the 82289.

Once the designer has determined which of the four bus-retention techniques to adopt, this configuration must be incorporated into the hardware design. Figure 7-11 shows the strapping configurations required to implement each of these four techniques.

Asserting the $\overline{\text{LOCK}}$ Signal

Independent of the particular operating mode of the Bus Arbiter, the iAPX 286 processor can assert a $\overline{\text{LOCK}}$ signal at any time to prevent the Bus Arbiter from releasing the Multibus. This software-controlled $\overline{\text{LOCK}}$ signal prevents the Bus Arbiter from surrendering the system bus to any other bus master, whether of higher- or lower-priority. This signal is typically used for implementing software semaphores for critical code sections or critical real-time events. The $\overline{\text{LOCK}}$ signal can also be asserted within one instruction to retain control of the bus for high-performance transfers.

When the iAPX 286 asserts the $\overline{\text{LOCK}}$ signal, the Bus Arbiter converts this temporary input into a level-lock signal, $\overline{\text{LLOCK}}$, which drives the Multibus $\overline{\text{LOCK}}$ status line. The Bus Arbiter will continue to assert $\overline{\text{LLOCK}}$ retaining control of the Multibus until the first unLOCKed bus cycle from the iAPX 286 processor.

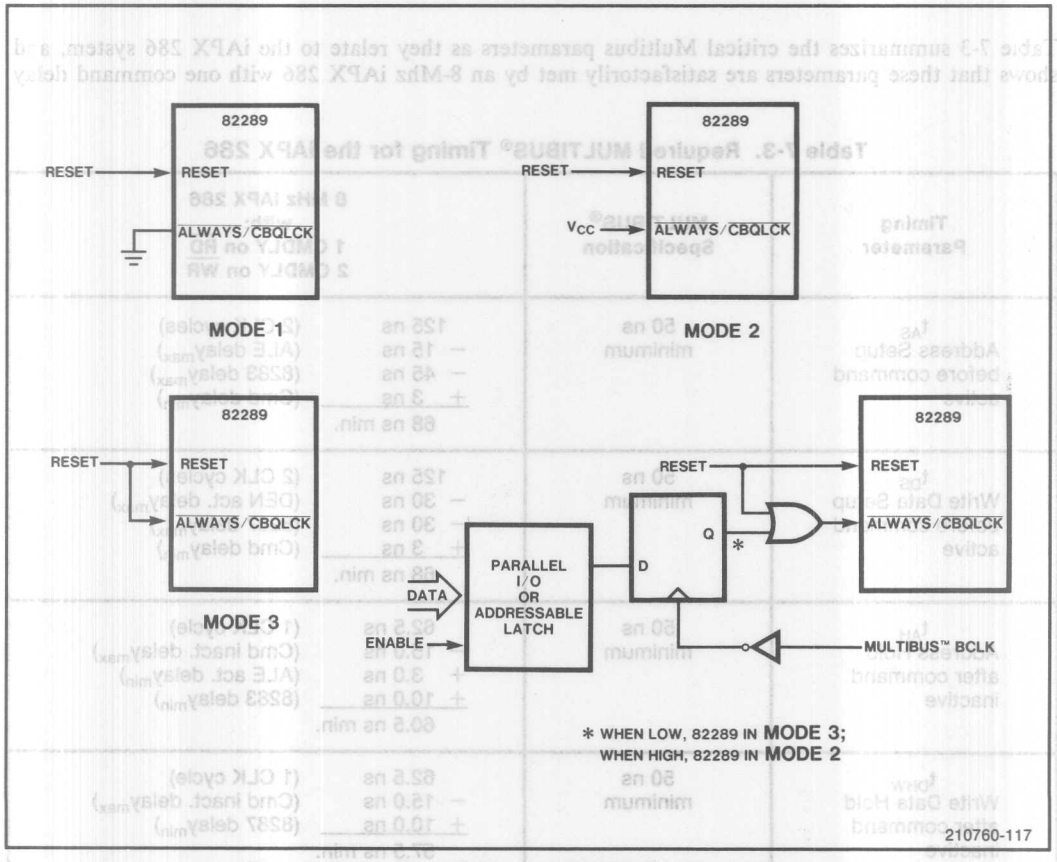


Figure 7-11. Four Different 82289 Configurations

This $\overline{\text{LLOCK}}$ signal from the 82289 Bus Arbiter must be connected to the Multibus $\overline{\text{LOCK}}$ status line through a tri-state driver. This driver is controlled by the AEN output of the Bus Arbiter.

TIMING ANALYSIS OF THE MULTIBUS® INTERFACE

The timing specifications for the Multibus are explained concisely in the Multibus specification, Order Number 9800683-04. To summarize these requirements as they pertain to an iAPX 286 subsystem operating as a Multibus bus master:

- The iAPX 286 system must use one command delay when reading data from the Multibus.
- The iAPX 286 must use two command delays when writing data to the Multibus.

When the 82288 Bus Controller driving the Multibus is strapped in the Multibus configuration (MB = 1, and CMDLY = 0), the Bus Controller automatically inserts the appropriate delays as outlined above. No further consideration is required in order to conform to the Multibus timing requirements.

Table 7-3 summarizes the critical Multibus parameters as they relate to the iAPX 286 system, and shows that these parameters are satisfactorily met by an 8-Mhz iAPX 286 with one command delay

Table 7-3. Required MULTIBUS® Timing for the iAPX 286

Timing Parameter	MULTIBUS® Specification	8 MHz iAPX 286 with: 1 CMDLY on $\overline{\text{RD}}$ 2 CMDLY on $\overline{\text{WR}}$	
t_{AS} Address Setup before command active	50 ns minimum	125 ns − 15 ns − 45 ns + 3 ns 68 ns min.	(2 CLK cycles) (ALE $delay_{max}$) (8283 $delay_{max}$) (Cmd $delay_{min}$)
t_{DS} Write Data Setup before command active	50 ns minimum	125 ns − 30 ns − 30 ns + 3 ns 68 ns min.	(2 CLK cycles) (DEN act. $delay_{max}$) (8287 $delay_{max}$) (Cmd $delay_{min}$)
t_{AH} Address Hold after command inactive	50 ns minimum	62.5 ns − 15.0 ns + 3.0 ns + 10.0 ns 60.5 ns min.	(1 CLK cycle) (Cmd inact. $delay_{max}$) (ALE act. $delay_{min}$) (8283 $delay_{min}$)
t_{DHW} Write Data Hold after command inactive	50 ns minimum	62.5 ns − 15.0 ns + 10.0 ns 57.5 ns min.	(1 CLK cycle) (Cmd inact. $delay_{max}$) (8287 $delay_{min}$)

during reads and two command delays during write operations. The timing parameters assume the use of 8283 and 8287 latches and transceivers.

In addition to the specific parameters defined in table 3-1, designers must be sure that:

- To ensure sufficient access time for the slave device, bus operations must not be terminated until an $\overline{\text{XACK}}$ (transfer acknowledge) is received from the slave device.
- Following an $\overline{\text{MRDC}}$ or an $\overline{\text{IORC}}$ command, the responding slave device must disable its data drivers within 125 ns (max) following the return of the $\overline{\text{XACK}}$ signal. (All devices meeting the Multibus spec. of 65 ns max meet this requirement.)

USING DUAL-PORT RAM WITH THE SYSTEM BUS

A dual-port RAM is a memory subsystem that can be accessed by the iAPX 286 via the iAPX 286 local bus, and can also be accessed by other processing elements via the Multibus system bus. The performance advantages of using dual-port memory are described in Chapter Four. This section describes several issues that must be considered when implementing dual-port memory with a Multibus interface.

Dual-port memories are a shared resource, and, like a shared bus, must address the twin issues of arbitration and mutual exclusion. Mutual exclusion addresses the case when two processors attempt to access the dual-port memory simultaneously. The Multibus $\overline{\text{LOCK}}$ signal and a similar $\overline{\text{LOCK}}$ signal from the 80286 itself attempt to mediate this contention.

Chapter Four of this manual describes the techniques of addressing these two issues. This section expands on the explanation in Chapter Four and in particular discusses how to handle the two $\overline{\text{LOCK}}$ signals and avoid potential deadlock conditions that may otherwise arise.

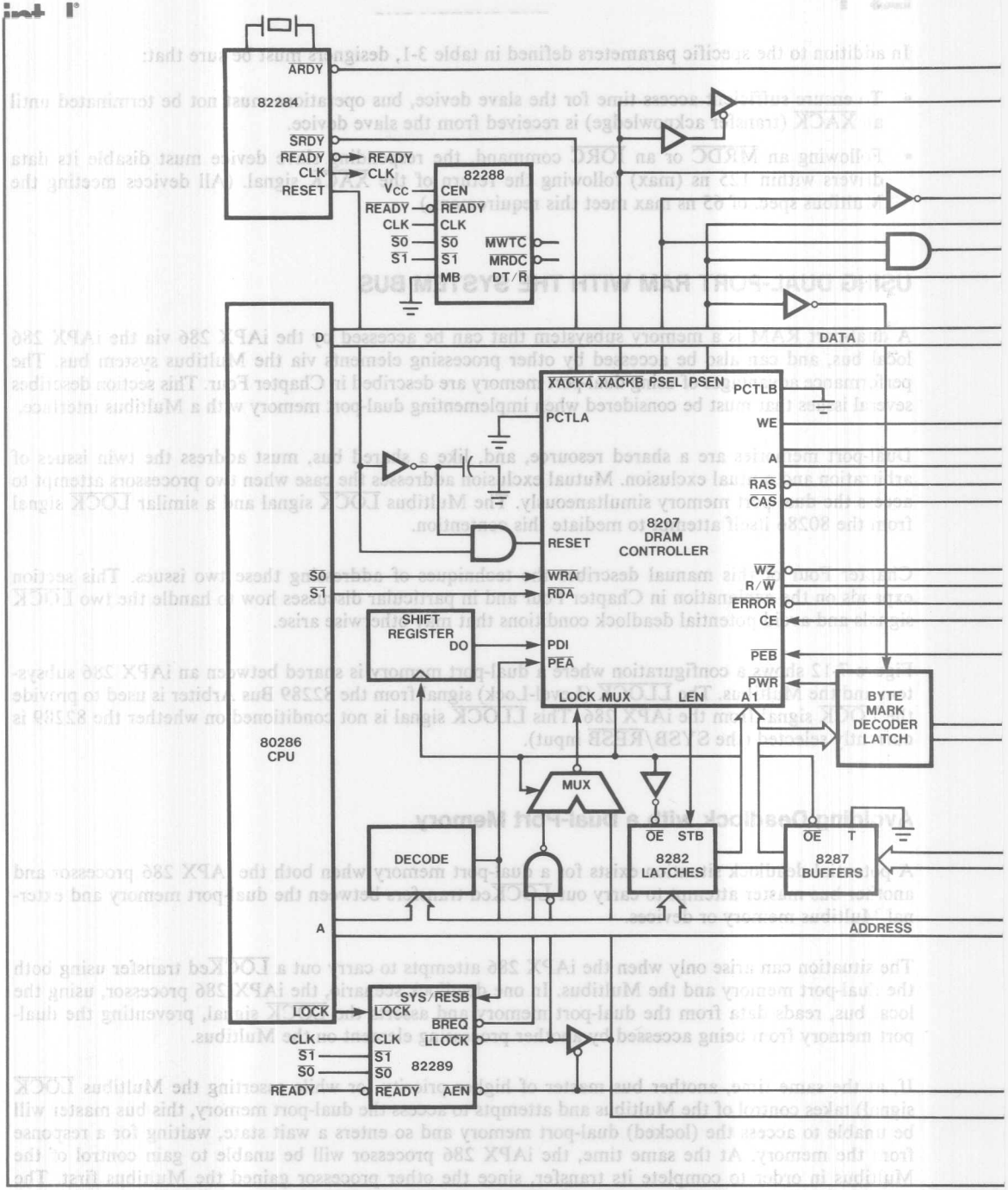
Figure 7-12 shows a configuration where a dual-port memory is shared between an iAPX 286 subsystem and the Multibus. The $\overline{\text{LLOCK}}$ (Level-Lock) signal from the 82289 Bus Arbiter is used to provide the $\overline{\text{LOCK}}$ signal from the iAPX 286. This $\overline{\text{LLOCK}}$ signal is not conditioned on whether the 82289 is currently selected (the SYSB/$\overline{\text{RESB}}$ input).

Avoiding Deadlock with a Dual-Port Memory

A potential deadlock situation exists for a dual-port memory when both the iAPX 286 processor and another bus master attempt to carry out $\overline{\text{LOCK}}$ed transfers between the dual-port memory and external Multibus memory or devices.

The situation can arise only when the iAPX 286 attempts to carry out a $\overline{\text{LOCK}}$ed transfer using both the dual-port memory and the Multibus. In one deadlock scenario, the iAPX 286 processor, using the local bus, reads data from the dual-port memory and asserts the $\overline{\text{LOCK}}$ signal, preventing the dual-port memory from being accessed by another processing element on the Multibus.

If, at the same time, another bus master of higher priority (or while asserting the Multibus $\overline{\text{LOCK}}$ signal) takes control of the Multibus and attempts to access the dual-port memory, this bus master will be unable to access the (locked) dual-port memory and so enters a wait state, waiting for a response from the memory. At the same time, the iAPX 286 processor will be unable to gain control of the Multibus in order to complete its transfer, since the other processor gained the Multibus first. The result is deadlock, and in the absence of other mechanisms, each processor stops.


ARDY
82284
SRDY
READY
CLK
RESET
READY
CLK
82288
CEN
READY
CLK
S0
S1
MB
VCC
MWTC
MRDC
DT/R
D
DATA
XACKA XACKB PSEL PSEN
PCTLB
PCTLA
WE
A
RAS
CAS
8207
DRAM
CONTROLLER
RESET
S0
S1
WRA
RDA
WZ
R/W
ERROR
CE
SHIFT
REGISTER
DO
PDI
PEA
PEB
PWR
LOCK MUX
LEN
A1
BYTE
MARK
DECODER
LATCH
80286
CPU
MUX
DECODE
OE
STB
8282
LATCHES
OE
T
8287
BUFFERS
A
ADDRESS
SYS/RESB
LOCK
BREQ
LLOCK
CLK
S1
S0
82289
READY
AEN

210760-001

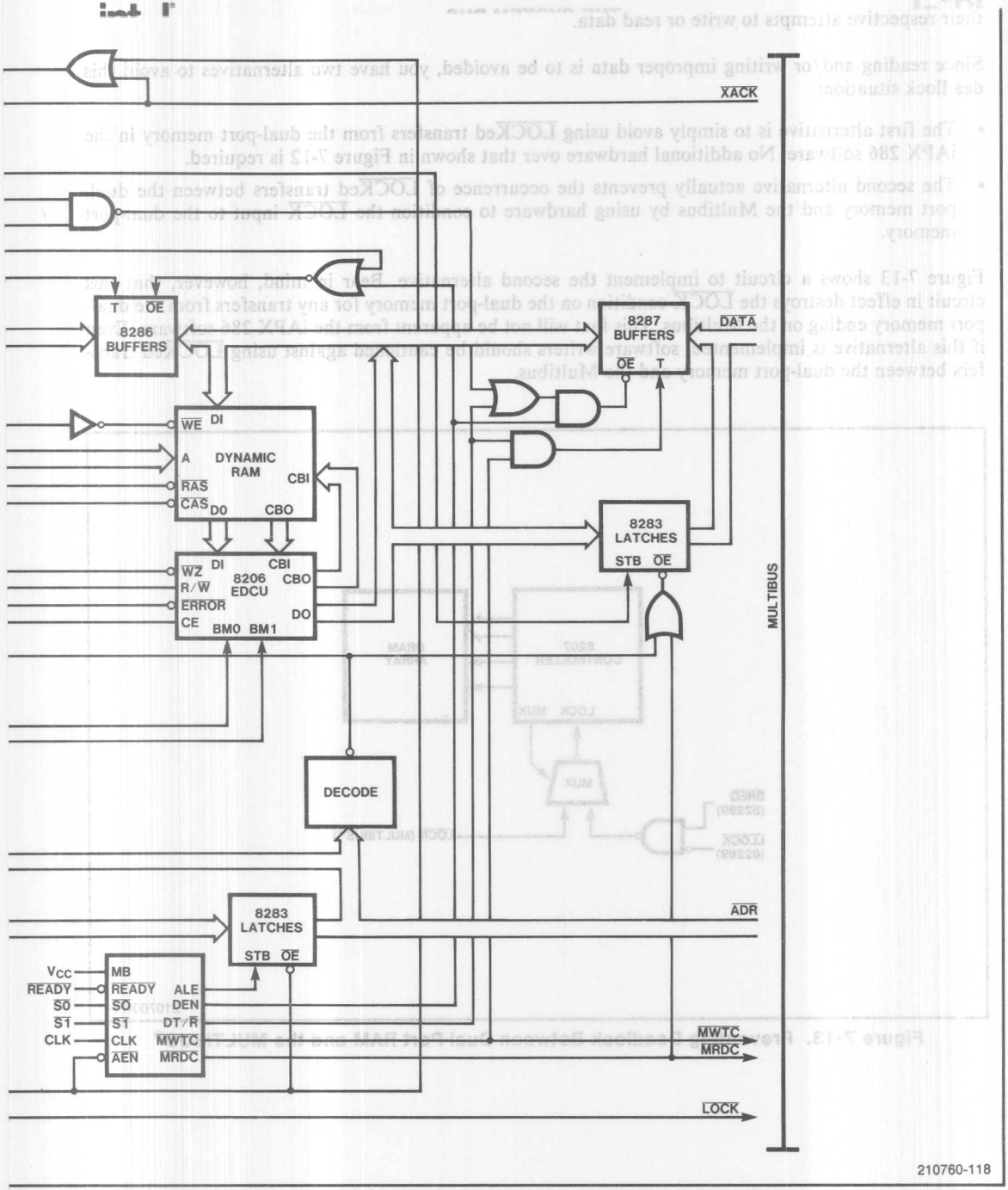


Figure 7-12. iAPX 286 Dual-Port Memory with MULTIBUS® Interface

Typically, both processors will eventually terminate their respective bus operations due to bus-timeout (see the earlier section on bus-timeout circuitry). If a bus timeout occurs, the processors will fail in their respective attempts to write or read data.

Since reading and/or writing improper data is to be avoided, you have two alternatives to avoid this deadlock situation:

- The first alternative is to simply avoid using $\overline{\text{LOCK}}$ed transfers from the dual-port memory in the iAPX 286 software. No additional hardware over that shown in Figure 7-12 is required.
- The second alternative actually prevents the occurrence of $\overline{\text{LOCK}}$ed transfers between the dual-port memory and the Multibus by using hardware to condition the $\overline{\text{LOCK}}$ input to the dual-port memory.

Figure 7-13 shows a circuit to implement the second alternative. Bear in mind, however, that this circuit in effect destroys the $\overline{\text{LOCK}}$ condition on the dual-port memory for any transfers from the dual-port memory ending on the Multibus. This fact will not be apparent from the iAPX 286 software. Even if this alternative is implemented, software writers should be cautioned against using $\overline{\text{LOCK}}$ed transfers between the dual-port memory and the Multibus.

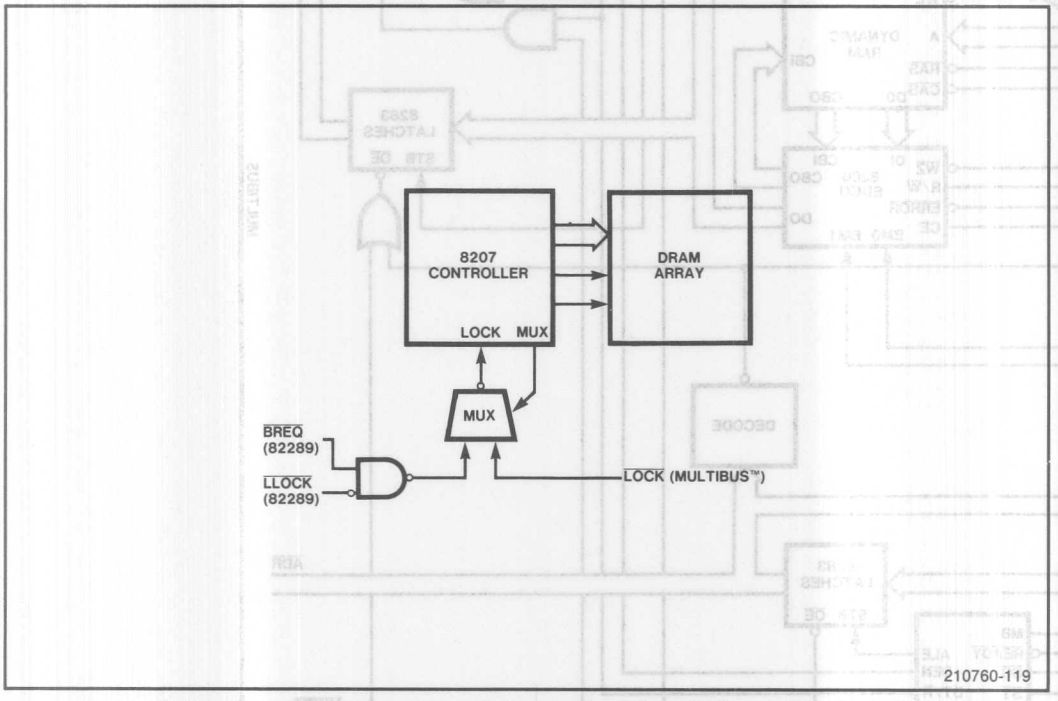


Figure 7-13. Preventing Deadlock Between Dual-Port RAM and the MULTIBUS®

Appendix A
Device Specifications

A

- **High Performance 8 and 10 MHz Processor (Up to six times iAPX 86)**
- **Large Address Space:**
 - **—16 Megabytes Physical**
 - **—1 Gigabyte Virtual per Task**
- **Integrated Memory Management, Four-Level Memory Protection and Support for Virtual Memory and Operating Systems**
- **Two iAPX 86 Upward Compatible Operating Modes:**
 - **—iAPX 86 Real Address Mode**
 - **—Protected Virtual Address Mode**
- **Optional Processor Extension:**
 - **—iAPX 286/20 High Performance 80-bit Numeric Data Processor**
- **Complete System Development Support:**
 - **—Development Software: Assembler, PL/M, Pascal, FORTRAN, and System Utilities**
 - **—In-Circuit-Emulator (ICE™-286)**
- **High Bandwidth Bus Interface (8 or 10 Megabyte/Sec)**
- **Available in EXPRESS:**
 - **—Standard Temperature Range**

The iAPX 286/10 (80286 part number) is an advanced, high-performance microprocessor with specially optimized capabilities for multiple user and multi-tasking systems. The 80286 has built-in memory protection that supports operating system and task isolation as well as program and data privacy within tasks. A 10 MHz iAPX 286/10 provides up to six times greater throughput than the standard 5 MHz iAPX 86/10. The 80286 includes memory management capabilities that map up to 2^{30} bytes (one gigabyte) of virtual address space per task into 2^{24} bytes (16 megabytes) of physical memory.

The iAPX 286 is upward compatible with iAPX 86 and 88 software. Using iAPX 86 real address mode, the 80286 is object code compatible with existing iAPX 86, 88 software. In protected virtual address mode, the 80286 is source code compatible with iAPX 86, 88 software and may require upgrading to use virtual addresses supported by the 80286's integrated memory management and protection mechanism. Both modes operate at full 80286 performance and execute a superset of the iAPX 86 and 88's instructions.

The 80286 provides special operations to support the efficient implementation and execution of operating systems. For example, one instruction can end execution of one task, save its state, switch to a new task, load its state, and start execution of the new task. The 80286 also supports virtual memory systems by providing a segment-not-present exception and restartable instructions.

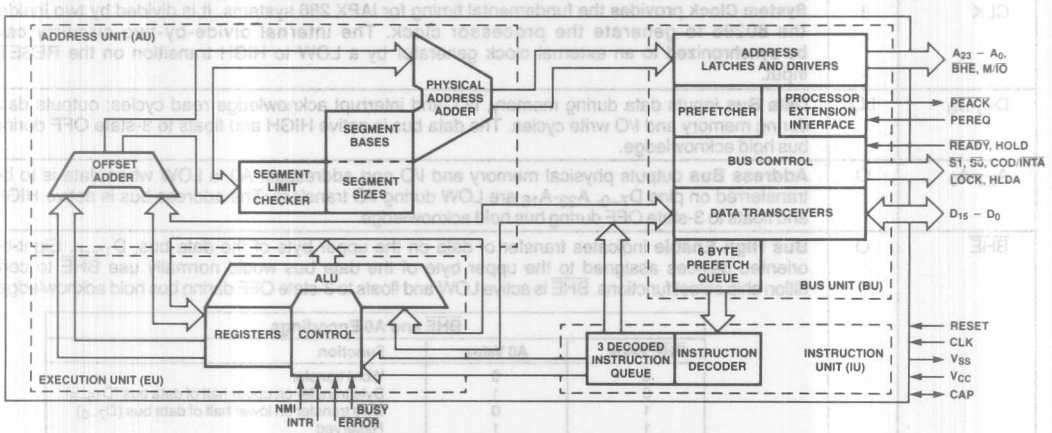


Figure 1. 80286 Internal Block Diagram

The following are trademarks of Intel Corporation and its affiliates and may be used only to identify Intel products: BXP, CREDIT, i, ICE, iCS, im, Insite, Intel, INTEL, Intelevision, Intellink, Intellec, iMMX, iOSP, iPDS, iRMX, iSBC, iSBX, Library Manager, MCS, MULTIMODULE, Megachassis, Micromainframe, MULTIBUS, Multichannel, Plug-A-Bubble, PROMPT, Promware, RUPI, RMX/80, System 2000, UPI, and the combination of iCS, iRMX, iSBC, iSBX, ICE, i²ICE, MCS, or UPI and a numerical suffix. Intel Corporation Assumes No Responsibility for the use of Any Circuitry Other Than Circuitry Embodied in an Intel Product. No Other Patent Licenses are implied. ©INTEL CORPORATION, 1982.
ORDER NUMBER: 210253-002

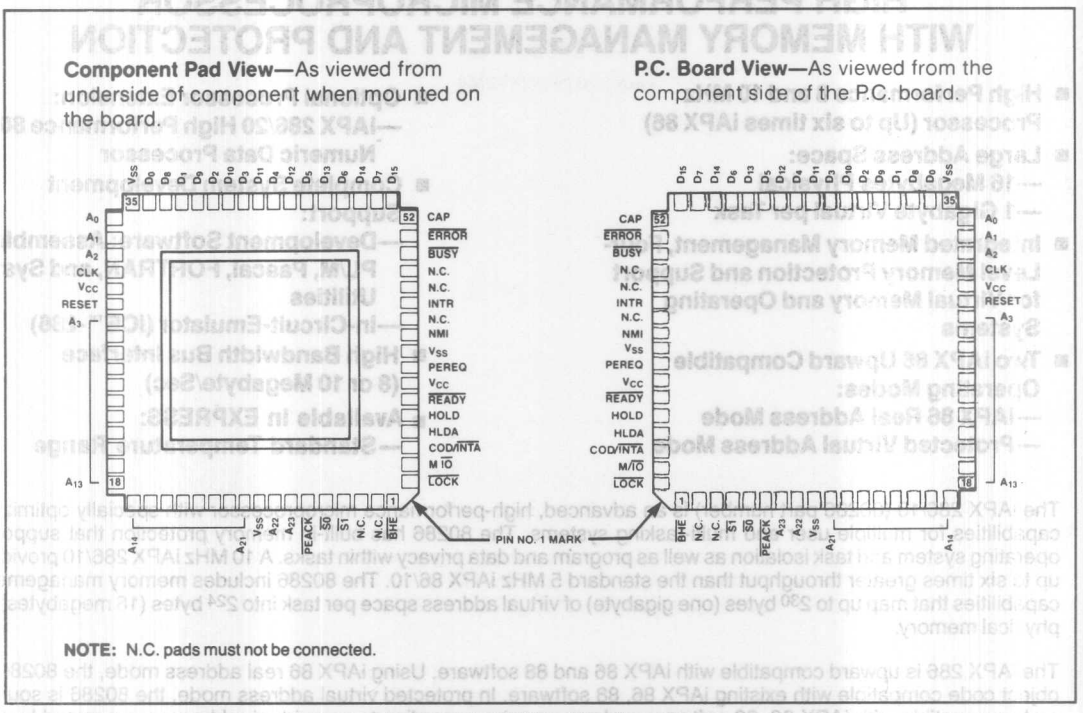


Figure 2. 80286 Pin Configuration

Table 1. Pin Description

The following pin function descriptions are for the 80286 microprocessor:

Symbol	Type	Name and Function
CLK	I	**System Clock** provides the fundamental timing for iAPX 286 systems. It is divided by two inside the 80286 to generate the processor clock. The internal divide-by-two circuitry can be synchronized to an external clock generator by a LOW to HIGH transition on the RESET input.
D_{15}-D_0	I/O	**Data Bus** inputs data during memory, I/O, and interrupt acknowledge read cycles; outputs data during memory and I/O write cycles. The data bus is active HIGH and floats to 3-state OFF during bus hold acknowledge.
A_{23}-A_0	O	**Address Bus** outputs physical memory and I/O port addresses. A0 is LOW when data is to be transferred on pins D_{7-0}. A_{23}-A_{16} are LOW during I/O transfers. The address bus is active HIGH and floats to 3-state OFF during bus hold acknowledge.
$\overline{BHE}$	O	**Bus High Enable** indicates transfer of data on the upper byte of the data bus, D_{15-8}. Eight-bit oriented devices assigned to the upper byte of the data bus would normally use $\overline{BHE}$ to condition chip select functions. $\overline{BHE}$ is active LOW and floats to 3-state OFF during bus hold acknowledge.

$\overline{BHE}$ and A0 Encodings

$\overline{BHE}$ Value	A0 Value	Function
0	0	Word transfer
0	1	Byte transfer on upper half of data bus (D_{15-8})
1	0	Byte transfer on lower half of data bus (D_{7-0})
1	1	Reserved

Table 1. Pin Description (Cont.)

Symbol	Type	Name and Function
$\overline{S1}$, $\overline{S0}$	O	**Bus Cycle Status** indicates initiation of a bus cycle and, along with M/$\overline{IO}$ and COD/$\overline{INTA}$, defines the type of bus cycle. The bus is in a T_S state whenever one or both are LOW. $\overline{S1}$ and $\overline{S0}$ are active LOW and float to 3-state OFF during bus hold acknowledge.

80286 Bus Cycle Status Definition

COD/$\overline{INTA}$	M/$\overline{IO}$	$\overline{S1}$	$\overline{S0}$	Bus cycle initiated
0 (LOW)	0	0	0	Interrupt acknowledge
0	0	0	1	Reserved
0	0	1	0	Reserved
0	0	1	1	None; not a status cycle
0	1	0	0	IF A1 = 1 then halt; else shutdown
0	1	0	1	Memory data read
0	1	1	0	Memory data write
0	1	1	1	None; not a status cycle
1 (HIGH)	0	0	0	Reserved
1	0	0	1	I/O read
1	0	1	0	I/O write
1	0	1	1	None; not a status cycle
1	1	0	0	Reserved
1	1	0	1	Memory instruction read
1	1	1	0	Reserved
1	1	1	1	None; not a status cycle

M/$\overline{IO}$	O	**Memory/IO Select** distinguishes memory access from I/O access. If HIGH during T_S, a memory cycle or a halt/shutdown cycle is in progress. If LOW, an I/O cycle or an interrupt acknowledge cycle is in progress. M/$\overline{IO}$ floats to 3-state OFF during bus hold acknowledge.
COD/$\overline{INTA}$	O	**Code/Interrupt Acknowledge** distinguishes instruction fetch cycles from memory data read cycles. Also distinguishes interrupt acknowledge cycles from I/O cycles. COD/$\overline{INTA}$ floats to 3-state OFF during bus hold acknowledge.
$\overline{LOCK}$	O	**Bus Lock** indicates that other system bus masters are not to gain control of the system bus following the current bus cycle. The $\overline{LOCK}$ signal may be activated explicitly by the "LOCK" instruction prefix or automatically by 80286 hardware during memory XCHG instructions, interrupt acknowledge, or descriptor table access. $\overline{LOCK}$ is active LOW and floats to 3-state OFF during bus hold acknowledge.
$\overline{READY}$	I	**Bus Ready** terminates a bus cycle. Bus cycles are extended without limit until terminated by $\overline{READY}$ LOW. $\overline{READY}$ is an active LOW synchronous input requiring setup and hold times relative to the system clock be met for correct operation. $\overline{READY}$ is ignored during bus hold acknowledge.
HOLD HLDA	I O	**Bus Hold Request and Hold Acknowledge** control ownership of the 80286 local bus. The HOLD input allows another local bus master to request control of the local bus. When control is granted, the 80286 will float its bus drivers to 3-state OFF and then activate HLDA, thus entering the bus hold acknowledge condition. The local bus will remain granted to the requesting master until HOLD becomes inactive which results in the 80286 deactivating HLDA and regaining control of the local bus. This terminates the bus hold acknowledge condition. HOLD may be asynchronous to the system clock. These signals are active HIGH.
INTR	I	**Interrupt Request** requests the 80286 to suspend its current program execution and service a pending external request. Interrupt requests are masked whenever the interrupt enable bit in the flag word is cleared. When the 80286 responds to an interrupt request, it performs two interrupt acknowledge bus cycles to read an 8-bit interrupt vector that identifies the source of the interrupt. To assure program interruption, INTR must remain active until the first interrupt acknowledge cycle is completed. INTR is sampled at the beginning of each processor cycle and must be active HIGH at least two processor cycles before the current instruction ends in order to interrupt before the next instruction. INTR is level sensitive, active HIGH, and may be asynchronous to the system clock.
NMI	I	**Non-maskable Interrupt Request** interrupts the 80286 with an internally supplied vector value of 2. No interrupt acknowledge cycles are performed. The interrupt enable bit in the 80286 flag word does not affect this input. The NMI input is active HIGH, may be asynchronous to the system clock, and is edge triggered after internal synchronization. For proper recognition, the input must have been previously LOW for at least four system clock cycles and remain HIGH for at least four system clock cycles.

PEREQ PEACK	I O	**Processor Extension Operand Request and Acknowledge** extend the memory management and protection capabilities of the 80286 to processor extensions. The PEREQ input requests the 80286 to perform a data operand transfer for a processor extension. The PEACK output signals the processor extension when the requested operand is being transferred. PEREQ is active HIGH and floats to 3-state OFF during bus hold acknowledge. PEACK may be asynchronous to the system clock. PEACK is active LOW.
BUSY ERROR	I I	**Processor Extension Busy and Error** indicate the operating condition of a processor extension to the 80286. An active BUSY input stops 80286 program execution on WAIT and some ESC instructions until BUSY becomes inactive (HIGH). The 80286 may be interrupted while waiting for BUSY to become inactive. An active ERROR input causes the 80286 to perform a processor extension interrupt when executing WAIT or some ESC instructions. These inputs are active LOW and may be asynchronous to the system clock.
RESET	I	**System Reset** clears the internal logic of the 80286 and is active HIGH. The 80286 may be re-initialized at any time with a LOW to HIGH transition on RESET which remains active for more than 16 system clock cycles. During RESET active, the output pins of the 80286 enter the state shown below: **80286 Pin State During Reset** **Pin Value** — **Pin Names** 1 (HIGH) — S0, S1, PEACK, A23-A0, BHE, LOCK 0 (LOW) — M/IO, COD/INTA, HLDA 3-state OFF — D_{15}–D_0 Operation of the 80286 begins after a HIGH to LOW transition on RESET. The HIGH to LOW transition of RESET must be synchronous to the system clock. Approximately 50 system clock cycles are required by the 80286 for internal initializations before the first bus cycle to fetch code from the power-on execution address is performed. A LOW to HIGH transition of RESET synchronous to the system clock will end a processor cycle at the second HIGH to LOW transition of the system clock. The LOW to HIGH transition of RESET may be asynchronous to the system clock; however, in this case it cannot be predetermined which phase of the processor clock will occur during the next system clock period. Synchronous LOW to HIGH transitions of RESET are required only for systems where the processor clock must be phase synchronous to another clock.
V_{SS}	I	**System Ground:** 0 VOLTS.
V_{CC}	I	**System Power:** +5 Volt Power Supply.
CAP	I	**Substrate Filter Capacitor:** a 0.047μf ± 20% 12V capacitor must be connected between this pin and ground. This capacitor filters the output of the internal substrate bias generator. A maximum DC leakage current of 1 μa is allowed through the capacitor. For correct operation of the 80286, the substrate bias generator must charge this capacitor to its operating voltage. The capacitor chargeup time is 5 milliseconds (max.) after V_{CC} and CLK reach their specified AC and DC parameters. RESET may be applied to prevent spurious activity by the CPU during this time. After this time, the 80286 processor clock can be phase synchronized to another clock by pulsing RESET LOW synchronous to the system clock.

FUNCTIONAL DESCRIPTION

Introduction

The 80286 is an advanced, high-performance microprocessor with specially optimized capabilities for multiple user and multi-tasking systems. Depending on the application, the 80286's performance is up to six times faster than the standard 5 MHz 8086's, while providing complete upward software compatibility with Intel's iAPX 86, 88, and 186 family of CPU's.

The 80286 operates in two modes: iAPX 86 real address mode and protected virtual address mode. Both modes execute a superset of the iAPX 86 and 88 instruction set.

In iAPX 86 real address mode programs use real addresses with up to one megabyte of address space. Programs use virtual addresses in protected virtual address mode, also called protected mode. In protected mode, the 80286 CPU automatically maps 1 gigabyte of virtual addresses per task into a 16 megabyte real address space. This mode also provides memory protection to isolate the operating system and ensure privacy of each tasks' programs and data. Both modes provide the same base instruction set, registers, and addressing modes.

The following Functional Description describes first, the base 80286 architecture common to both modes, second, iAPX 86 real address mode, and third, protected mode.

iAPX 286/10 BASE ARCHITECTURE

The iAPX 86, 88, 186, and 286 CPU family all contain the same basic set of registers, instructions, and addressing modes. The 80286 processor is upward compatible with the 8086, 8088, and 80186 CPU's.

Register Set

The 80286 base architecture has fifteen registers as shown in Figure 3. These registers are grouped into the following four categories:

General Registers: Eight 16-bit general purpose registers used to contain arithmetic and logical operands. Four of these (AX, BX, CX, and DX) can be used either in their entirety as 16-bit words or split into pairs of separate 8-bit registers.

Segment Registers: Four 16-bit special purpose registers select, at any given time, the segments of memory that are immediately addressable for code, stack, and data. (For usage, refer to Memory Organization.)

Base and Index Registers: Four of the general purpose registers may also be used to determine offset addresses of operands in memory. These registers may contain base addresses or indexes to particular locations within a segment. The addressing mode determines the specific registers used for operand address calculations.

Status and Control Registers: The 3 16-bit special purpose registers in figure 3A record or control certain aspects of the 80286 processor state including the Instruction Pointer, which contains the offset address of the next sequential instruction to be executed.

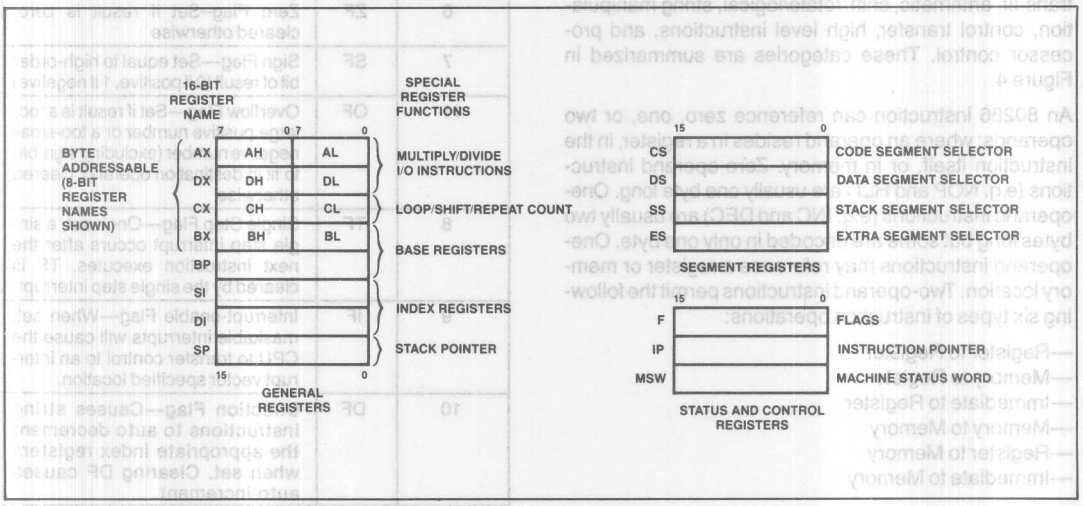


Figure 3. Register Set

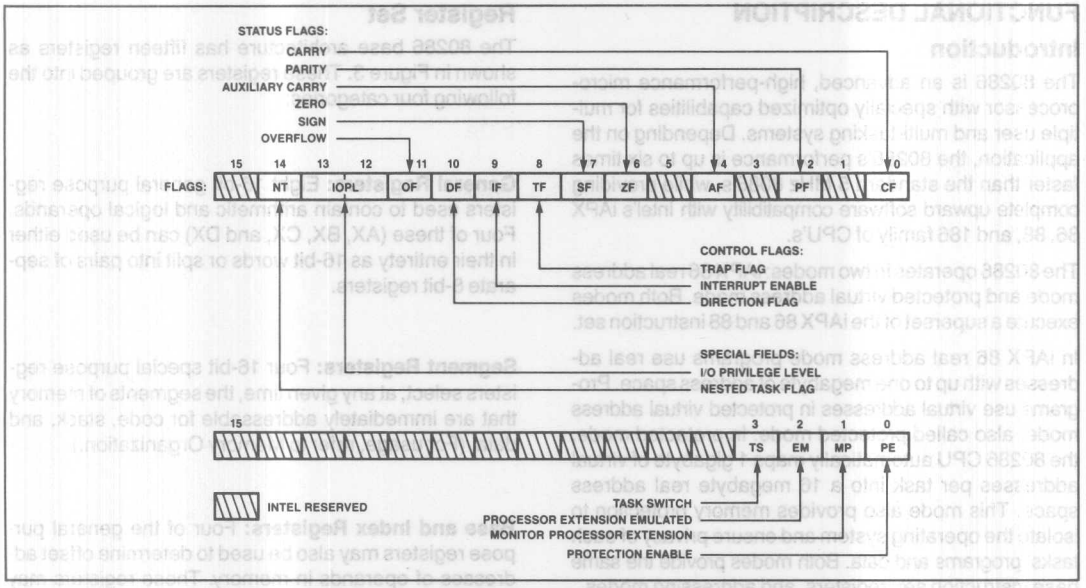


Figure 3a. Status and Control Register Bit Functions

Flags Word Description

The Flags word (Flags) records specific characteristics of the result of logical and arithmetic instructions (bits 0, 2, 4, 6, 7, and 11) and controls the operation of the 80286 within a given operating mode (bits 8 and 9). Flags is a 16-bit register. The function of the flag bits is given in Table 2.

Instruction Set

The instruction set is divided into seven categories: data transfer, arithmetic, shift/rotate/logical, string manipulation, control transfer, high level instructions, and processor control. These categories are summarized in Figure 4.

An 80286 instruction can reference zero, one, or two operands; where an operand resides in a register, in the instruction itself, or in memory. Zero-operand instructions (e.g. NOP and HLT) are usually one byte long. One-operand instructions (e.g. INC and DEC) are usually two bytes long but some are encoded in only one byte. One-operand instructions may reference a register or memory location. Two-operand instructions permit the following six types of instruction operations:

—Register to Register
—Memory to Register
—Immediate to Register
—Memory to Memory
—Register to Memory
—Immediate to Memory

Table 2. Flags Word Bit Functions

Bit Position	Name	Function
0	CF	Carry Flag—Set on high-order bit carry or borrow; cleared otherwise
2	PF	Parity Flag—Set if low-order 8 bits of result contain an even number of 1-bits; cleared otherwise
4	AF	Set on carry from or borrow to the low order four bits of AL; cleared otherwise
6	ZF	Zero Flag–Set if result is zero; cleared otherwise
7	SF	Sign Flag—Set equal to high-order bit of result (0 if positive, 1 if negative)
11	OF	Overflow Flag—Set if result is a too-large positive number or a too-small negative number (excluding sign-bit) to fit in destination operand; cleared otherwise
8	TF	Single Step Flag—Once set, a single step interrupt occurs after the next instruction executes. TF is cleared by the single step interrupt.
9	IF	Interrupt-enable Flag—When set, maskable interrupts will cause the CPU to transfer control to an interrupt vector specified location.
10	DF	Direction Flag—Causes string instructions to auto decrement the appropriate index registers when set. Clearing DF causes auto increment.

Two-operand instructions (e.g. MOV and ADD) are usually three to six bytes long. Memory to memory operations are provided by a special class of string instructions requiring one to three bytes. For detailed instruction formats and encodings refer to the instruction set summary at the end of this document.

GENERAL PURPOSE	
MOV	Move byte or word
PUSH	Push word onto stack
POP	Pop word off stack
PUSHA	Push all registers on stack
POPA	Pop all registers from stack
XCHG	Exchange byte or word
XLAT	Translate byte
INPUT/OUTPUT	
IN	Input byte or word
OUT	Output byte or word
ADDRESS OBJECT	
LEA	Load effective address
LDS	Load pointer using DS
LES	Load pointer using ES
FLAG TRANSFER	
LAHF	Load AH register from flags
SAHF	Store AH register in flags
PUSHF	Push flags onto stack
POPF	Pop flags off stack

Figure 4a. Data Transfer Instructions

MOVS	Move byte or word string
INS	Input bytes or word string
OUTS	Output bytes or word string
CMPS	Compare byte or word string
SCAS	Scan byte or word string
LODS	Load byte or word string
STOS	Store byte or word string
REP	Repeat
REPE/REPZ	Repeat while equal/zero
REPNE/REPNZ	Repeat while not equal/not zero

Figure 4c. String Instructions

ADDITION	
ADD	Add byte or word
ADC	Add byte or word with carry
INC	Increment byte or word by 1
AAA	ASCII adjust for addition
DAA	Decimal adjust for addition
SUBTRACTION	
SUB	Subtract byte or word
SBB	Subtract byte or word with borrow
DEC	Decrement byte or word by 1
NEG	Negate byte or word
CMP	Compare byte or word
AAS	ASCII adjust for subtraction
DAS	Decimal adjust for subtraction
MULTIPLICATION	
MUL	Multiply byte or word unsigned
IMUL	Integer multiply byte or word
AAM	ASCII adjust for multiply
DIVISION	
DIV	Divide byte or word unsigned
IDIV	Integer divide byte or word
AAD	ASCII adjust for division
CBW	Convert byte to word
CWD	Convert word to doubleword

Figure 4b. Arithmetic Instructions

LOGICALS	
NOT	"Not" byte or word
AND	"And" byte or word
OR	"Inclusive or" byte or word
XOR	"Exclusive or" byte or word
TEST	"Test" byte or word
SHIFTS	
SHL/SAL	Shift logical/arithmetic left byte or word
SHR	Shift logical right byte or word
SAR	Shift arithmetic right byte or word
ROTATES	
ROL	Rotate left byte or word
ROR	Rotate right byte or word
RCL	Rotate through carry left byte or word
RCR	Rotate through carry right byte or word

Figure 4d. Shift/Rotate/Logical Instructions

JA/JNBE	Jump if above/not below nor equal
JAE/JNB	Jump if above or equal/not below
JB/JNAE	Jump if below/not above nor equal
JBE/JNA	Jump if below or equal/not above
JC	Jump if carry
JE/JZ	Jump if equal/zero
JG/JNLE	Jump if greater/not less nor equal
JGE/JNL	Jump if greater or equal/not less
JL/JNGE	Jump if less/not greater nor equal
JLE/JNG	Jump if less or equal/not greater
JNC	Jump if not carry
JNE/JNZ	Jump if not equal/not zero
JNO	Jump if not overflow
JNP/JPO	Jump if not parity/parity odd
JNS	Jump if not sign
JO	Jump if overflow
JP/JPE	Jump if parity/parity even
JS	Jump if sign
CALL	Call procedure
RET	Return from procedure
JMP	Jump
ITERATION CONTROLS	
LOOP	Loop
LOOPE/LOOPZ	Loop if equal/zero
LOOPNE/LOOPNZ	Loop if not equal/not zero
JCXZ	Jump if register CX = 0
INTERRUPTS	
INT	Interrupt
INTO	Interrupt if overflow
IRET	Interrupt return

Figure 4e. Program Transfer Instructions

FLAG OPERATIONS	
STC	Set carry flag
CLC	Clear carry flag
CMC	Complement carry flag
STD	Set direction flag
CLD	Clear direction flag
STI	Set interrupt enable flag
CLI	Clear interrupt enable flag
EXTERNAL SYNCHRONIZATION	
HLT	Halt until interrupt or reset
WAIT	Wait for $\overline{\text{BUSY}}$ not active
ESC	Escape to extension processor
LOCK	Lock bus during next instruction
NO OPERATION	
NOP	No operation
EXECUTION ENVIRONMENT CONTROL	
LMSW	Load machine status word
SMSW	Store machine status word

Figure 4f. Processor Control Instructions

ENTER	Format stack for procedure entry
LEAVE	Restore stack for procedure exit
BOUND	Detects values outside prescribed range

Figure 4g. High Level Instructions

Memory Organization

Memory is organized as sets of variable length segments. Each segment is a linear contiguous sequence of up to 64K (2^{16}) 8-bit bytes. Memory is addressed using a two-component address (a pointer) that consists of a 16-bit segment selector, and a 16-bit offset. The segment selector indicates the desired segment in memory. The offset component indicates the desired byte address within the segment.

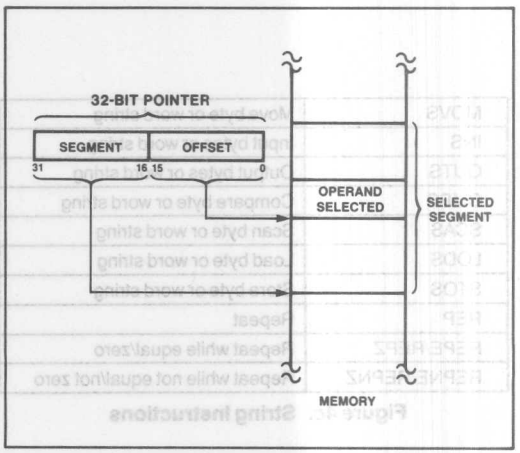


Figure 5. Two Component Address

Table 3. Segment Register Selection Rules

Memory Reference Needed	Segment Register Used	Implicit Segment Selection Rule
Instructions	Code (CS)	Automatic with instruction prefetch
Stack	Stack (SS)	All stack pushes and pops. Any memory reference which uses BP as a base register.
Local Data	Data (DS)	All data references except when relative to stack or string destination
External (Global) Data	Extra (ES)	Alternate data segment and destination of string operation

All instructions that address operands in memory must specify the segment and the offset. For speed and compact instruction encoding, segment selectors are usually stored in the high speed segment registers. An instruction need specify only the desired segment register and an offset in order to address a memory operand.

Most instructions need not explicitly specify which segment register is used. The correct segment register is automatically chosen according to the rules of Table 3. These rules follow the way programs are written (see Figure 6) as independent modules that require areas for code and data, a stack, and access to external data areas.

Special segment override instruction prefixes allow the implicit segment register selection rules to be overridden for special cases. The stack, data, and extra segments may coincide for simple programs. To access operands not residing in one of the four immediately available segments, a full 32-bit pointer or a new segment selector must be loaded.

Addressing Modes

The 80286 provides a total of eight addressing modes for instructions to specify operands. Two addressing modes are provided for instructions that operate on register or immediate operands:

Register Operand Mode: The operand is located in one of the 8 or 16-bit general registers.

Immediate Operand Mode. The operand is included in the instruction.

Six modes are provided to specify the location of an operand in a memory segment. A memory operand address consists of two 16-bit components: segment selector and offset. The segment selector is supplied by a segment register either implicitly chosen by the addressing mode or explicitly chosen by a segment override prefix. The offset is calculated by summing any combination of the following three address elements:

the **displacement** (an 8 or 16-bit immediate value contained in the instruction)

the **base** (contents of either the BX or BP base registers)

the **index** (contents of either the SI or DI index registers)

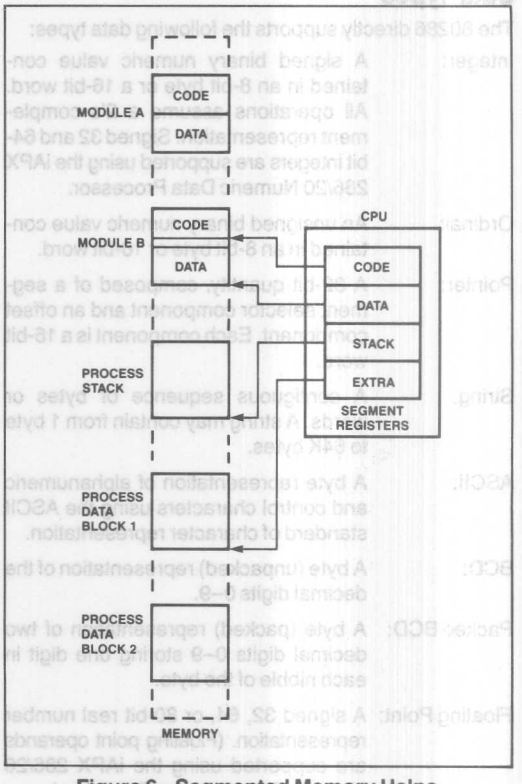


Figure 6. Segmented Memory Helps Structure Software

Any carry out from the 16-bit addition is ignored. Eight-bit displacements are sign extended to 16-bit values.

Combinations of these three address elements define the six memory addressing modes, described below.

Direct Mode: The operand's offset is contained in the instruction as an 8 or 16-bit displacement element.

Register Indirect Mode: The operand's offset is in one of the registers SI, DI, BX, or BP.

Based Mode: The operand's offset is the sum of an 8 or 16-bit displacement and the contents of a base register (BX or BP).

Indexed Mode: The operand's offset is the sum of an 8 or 16-bit displacement and the contents of an index register (SI or DI).

Based Mode: The operand's offset is the sum of the contents of a base register and an index register.

Based Indexed Mode with Displacement: The operand's offset is the sum of a base register's contents, an index register's contents, and an 8 or 16-bit displacement.

Data Types

The 80286 directly supports the following data types:

Integer: A signed binary numeric value contained in an 8-bit byte or a 16-bit word. All operations assume a 2's complement representation. Signed 32 and 64-bit integers are supported using the iAPX 286/20 Numeric Data Processor.

Ordinal: An unsigned binary numeric value contained in an 8-bit byte or 16-bit word.

Pointer: A 32-bit quantity, composed of a segment selector component and an offset component. Each component is a 16-bit word.

String: A contiguous sequence of bytes or words. A string may contain from 1 byte to 64K bytes.

ASCII: A byte representation of alphanumeric and control characters using the ASCII standard of character representation.

BCD: A byte (unpacked) representation of the decimal digits 0–9.

Packed BCD: A byte (packed) representation of two decimal digits 0–9 storing one digit in each nibble of the byte.

Floating Point: A signed 32, 64, or 80-bit real number representation. (Floating point operands are supported using the iAPX 286/20 Numeric Processor configuration.)

Figure 7 graphically represents the data types supported by the iAPX 286.

I/O Space

The I/O space consists of 64K 8-bit or 32K 16-bit ports. I/O instructions address the I/O space with either an 8-bit port address, specified in the instruction, or a 16-bit port address in the DX register. 8-bit port addresses are zero extended such that A_{15}–A_8 are LOW. I/O port addresses 00F8(H) through 00FF(H) are reserved.

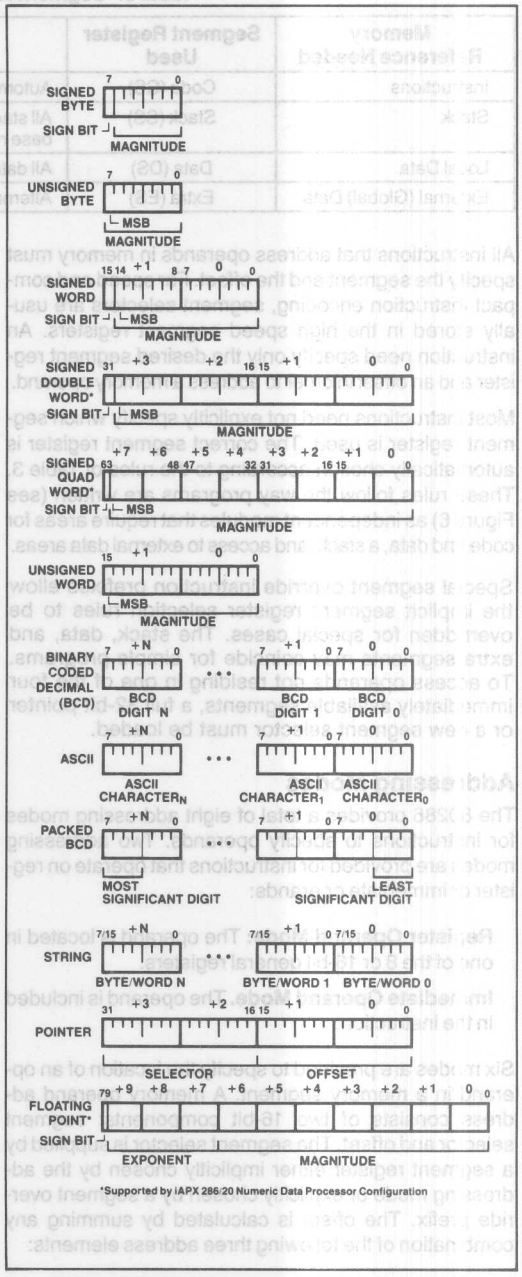


Figure 7. iAPX 286 Supported Data Types

Table 4. Interrupt Vector Assignments

Function	Interrupt Number	Related Instructions	Return Address Before Instruction Causing Exception?
Divide error exception	0	DIV, IDIV	Yes
Single step interrupt	1	All	
NMI interrupt	2	All	
Breakpoint interrupt	3	INT	
INTO detected overflow exception	4	INTO	No
BOUND range exceeded exception	5	BOUND	Yes
Invalid opcode exception	6	Any undefined opcode	Yes
Processor extension not available exception	7	ESC or WAIT	Yes
Reserved	8–15		
Processor extension error interrupt	16	ESC or WAIT	
Reserved	17–31		
User defined	32–255		

Interrupts

An interrupt transfers execution to a new program location. The old program address (CS:IP) and machine state (Flags) are saved on the stack to allow resumption of the interrupted program. Interrupts fall into three classes: hardware initiated, INT instructions, and instruction exceptions. Hardware initiated interrupts occur in response to an external input and are classified as non-maskable or maskable. Programs may cause an interrupt with an INT instruction. Instruction exceptions occur when an unusual condition, which prevents further instruction processing, is detected while attempting to execute an instruction. The return address from an exception will always point at the instruction causing the exception and include any leading instruction prefixes.

A table containing up to 256 pointers defines the proper interrupt service routine for each interrupt. Interrupts 0–31, some of which are used for instruction exceptions, are reserved. For each interrupt, an 8-bit vector must be supplied to the 80286 which identifies the appropriate table entry. Exceptions supply the interrupt vector internally. INT instructions contain or imply the vector and allow access to all 256 interrupts. Maskable hardware initiated interrupts supply the 8-bit vector to the CPU during an interrupt acknowledge bus sequence. Non-maskable hardware interrupts use a predefined internally supplied vector.

MASKABLE INTERRUPT (INTR)

The 80286 provides a maskable hardware interrupt request pin, INTR. Software enables this input by setting the interrupt flag bit (IF) in the flag word. All 224 user-defined interrupt sources can share this input, yet they can retain separate interrupt handlers. An 8-bit vector read by the CPU during the interrupt acknowledge sequence (discussed in System Interface section) identifies the source of the interrupt.

Further maskable interrupts are disabled while servicing an interrupt by resetting the IF but as part of the response to an interrupt or exception. The saved flag word will reflect the enable status of the processor prior to the interrupt. Until the flag word is restored to the flag register, the interrupt flag will be zero unless specifically set. The interrupt return instruction includes restoring the flag word, thereby restoring the original status of IF.

NON-MASKABLE INTERRUPT REQUEST (NMI)

A non-maskable interrupt input (NMI) is also provided. NMI has higher priority than INTR. A typical use of NMI would be to activate a power failure routine. The activation of this input causes an interrupt with an internally supplied vector value of 2. No external interrupt acknowledge sequence is performed.

While executing the NMI servicing procedure, the 80286 will service neither further NMI requests, INTR requests, nor the processor extension segment overrun interrupt until an interrupt return (IRET) instruction is executed or the CPU is reset. If NMI occurs while currently servicing an NMI, its presence will be saved for servicing after executing the first IRET instruction. IF is cleared at the beginning of an NMI interrupt to inhibit INTR interrupts.

The 80286 has an internal interrupt that allows programs to execute one instruction at a time. It is called the single step interrupt and is controlled by the single step flag bit (TF) in the flag word. Once this bit is set, an internal single step interrupt will occur after the next instruction has been executed. The interrupt clears the TF bit and uses an internally supplied vector of 1. The IRET instruction is used to set the TF bit and transfer control to the next instruction to be single stepped.

Interrupt Priorities

When simultaneous interrupt requests occur, they are processed in a fixed order as shown in Table 5. Interrupt processing involves saving the flags, return address, and setting CS:IP to point at the first instruction of the interrupt handler. If other interrupts remain enabled they are processed before the first instruction of the current interrupt handler is executed. The last interrupt processed is therefore the first one serviced.

Table 5. Interrupt Processing Order

Order	Interrupt
1	INT instruction or exception
2	Single step
3	NMI
4	Processor extension segment overrun
5	INTR

Initialization and Processor Reset

Processor initialization or start up is accomplished by driving the RESET input pin HIGH. RESET forces the 80286 to terminate all execution and local bus activity. No instruction or bus activity will occur as long as RESET is active. After RESET becomes inactive and an internal processing interval elapses, the 80286 begins execution in real address mode with the instruction at physical location FFFFF0(H). RESET also sets some registers to predefined values as shown as shown in Table 6 .

Flag word	0002(H)
Machine Status Word	FFF0(H)
Instruction pointer	FFF0(H)
Code segment	F000(H)
Data segment	0000(H)
Extra segment	0000(H)
Stack segment	0000(H)

Machine Status Word Description

The machine status word (MSW) records when a task switch takes place and controls the operating mode of the 80286. It is a 16-bit register of which the lower four bits are used. One bit places the CPU into protected mode, while the other three bits, as shown in Table 7, control the processor extension interface. After RESET, this register contains FFF0(H) which places the 80286 in iAPX 86 real address mode.

Table 7. MSW Bit Functions

Bit Position	Name	Function
0	PE	Protected mode enable places the 80286 into protected mode and can not be cleared except by RESET.
1	MP	Monitor processor extension allows WAIT instructions to cause a processor extension not present exception (number 7).
2	EM	Emulate processor extension causes a processor extension not present exception (number 7) on ESC instructions to allow emulating a processor extension.
3	TS	Task switched indicates the next instruction using a processor extension will cause exception 7, allowing software to test whether the current processor extension context belongs to the current task.

The LMSW and SMSW instructions can load and store the MSW in real address mode. The recommended use of TS, EM, and MP is shown in Table 8.

Table 8. Recommended MSW Encodings For Processor Extension Control

TS	MP	EM	Recommended Use	Instructions Causing Exception 7
0	0	0	Initial encoding after RESET. iAPX 286 operation is identical to iAPX 86,88.	None
0	0	1	No processor extension is available. Software will emulate its function.	ESC
1	0	1	No processor extension is available. Software will emulate its function. The current processor extension context may belong to another task.	ESC
0	1	0	A processor extension exists.	None
1	1	0	A processor extension exists. The current processor extension context may belong to another task. The exception on WAIT allows software to test for an error pending from a previous processor extension operation.	ESC or WAIT

Halt

The HLT instruction stops program execution and prevents the CPU from using the local bus until restarted. Either NMI, INTR with IF = 1, or RESET will force the 80286 out of halt. If interrupted, the saved CS:IP will point to the next instruction after the HLT.

iAPX 86 REAL ADDRESS MODE

The 80286 executes a fully upward-compatible superset of the 8086 instruction set in real address mode. In real address mode the 80286 is object code compatible with 8086 and 8088 software. The real address mode architecture (registers and addressing modes) is exactly as described in the iAPX 286/10 Base Architecture section of this Functional Description.

Memory Size

Physical memory is a contiguous array of up to 1,048,576 bytes (one megabyte) addressed by pins A_0 through A_{19} and $\overline{BHE}$. A_{20} through A_{23} may be ignored.

Memory Addressing

In real address mode the processor generates 20-bit physical addresses directly from a 20-bit segment base address and a 16-bit offset.

The selector portion of a pointer is interpreted as the upper 16 bits of a 20-bit segment address. The lower four bits of the 20-bit segment address are always zero. Segment addresses, therefore, begin on multiples of 16 bytes. See Figure 8 for a graphic representation of address formation.

All segments in real address mode are 64K bytes in size and may be read, written, or executed. An exception or interrupt can occur if data operands or instructions attempt to wrap around the end of a segment (e.g. a word with its low order byte at offset FFFF(H) and its high order byte at offset 0000(H)). If, in real address mode, the information contained in a segment does not use the full 64K bytes, the unused end of the segment may be overlayed by another segment to reduce physical memory requirements.

Reserved Memory Locations

The 80286 reserves two fixed areas of memory in real address mode (see Figure 9); system initialization area and interrupt table area. Locations from addresses FFFF0(H) thorugh FFFFF(H) are reserved for system initialization. Initial execution begins at location FFFF0(H). Locations 00000(H) through 003FF(H) are reserved for interrupt vectors.

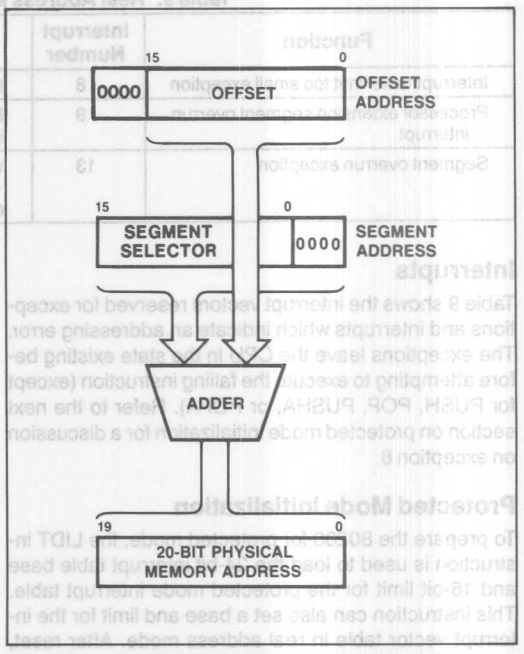


Figure 8. iAPX 86 Real Address Mode Address Calculation

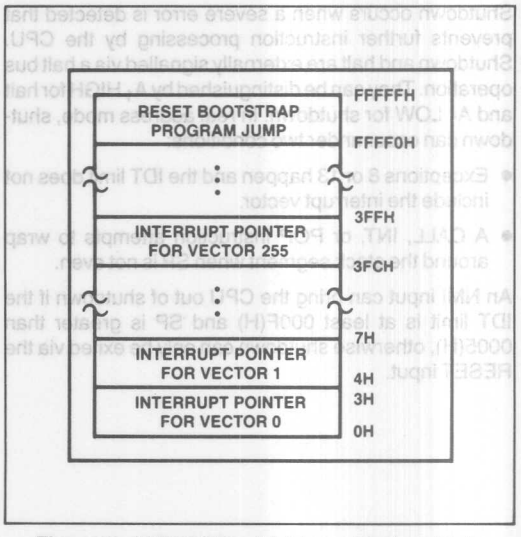


Figure 9. iAPX 86 Real Address Mode Initially Reserved Memory Locations

Table 9. Real Address Mode Addressing Interrupts

Function	Interrupt Number	Related Instructions	Return Address Before Instruction?
Interrupt table limit too small exception	8	INT vector is not within table limit	Yes
Processor extension segment overrun interrupt	9	ESC with memory operand extending beyond offset FFFF(H)	No
Segment overrun exception	13	Word memory reference with offset = FFFF(H) or an attempt to execute past the end of a segment	Yes

Interrupts

Table 9 shows the interrupt vectors reserved for exceptions and interrupts which indicate an addressing error. The exceptions leave the CPU in the state existing before attempting to execute the failing instruction (except for PUSH, POP, PUSHA, or POPA). Refer to the next section on protected mode initialization for a discussion on exception 8.

Protected Mode Initialization

To prepare the 80286 for protected mode, the LIDT instruction is used to load the 24-bit interrupt table base and 16-bit limit for the protected mode interrupt table. This instruction can also set a base and limit for the interrupt vector table in real address mode. After reset, the interrupt table base is initialized to 000000(H) and its size set to 03FF(H). These values are compatible with iAPX 86, 88 software. LIDT should only be executed in preparation for protected mode.

Shutdown

Shutdown occurs when a severe error is detected that prevents further instruction processing by the CPU. Shutdown and halt are externally signalled via a halt bus operation. They can be distinguished by A_1 HIGH for halt and A_1 LOW for shutdown. In real address mode, shutdown can occur under two conditions:

- Exceptions 8 or 13 happen and the IDT limit does not include the interrupt vector.
- A CALL, INT, or POP instruction attempts to wrap around the stack segment when SP is not even.

An NMI input can bring the CPU out of shutdown if the IDT limit is at least 000F(H) and SP is greater than 0005(H), otherwise shutdown can only be exited via the RESET input.

PROTECTED VIRTUAL ADDRESS MODE

The 80286 executes a fully upward-compatible superset of the 8086 instruction set in protected virtual address mode (protected mode). Protected mode also provides memory management and protection mechanisms and associated instructions.

The 80286 enters protected virtual address mode from real address mode by setting the PE (Protection Enable) bit of the machine status word with the Load Machine Status Word (LMSW) instruction. Protected mode offers extended physical and virtual memory address space, memory protection mechanisms, and new operations to support operating systems and virtual memory.

All registers, instructions, and addressing modes described in the iAPX 286/10 Base Architecture section of this Functional Description remain the same. Programs for the iAPX 86, 88, 186, and real address mode 80286 can be run in protected mode; however, embedded constants for segment selectors are different.

Memory Size

The protected mode 80286 provides a 1 gigabyte virtual address space per task mapped into a 16 megabyte physical address space defined by the address pins A_{23}–A_0 and $\overline{BHE}$. The virtual address space may be larger than the physical address space since any use of an address that does not map to a physical memory location will cause a restartable exception.

Memory Addressing

As in real address mode, protected mode uses 32-bit pointers, consisting of 16-bit selector and offset components. The selector, however, specifies an index into a memory resident table rather than the upper 16-bits of a real memory address. The 24-bit base address of the

desired segment is obtained from the tables in memory. The 16-bit offset is added to the segment base address to form the physical address as shown in Figure 10. The tables are automatically referenced by the CPU whenever a segment register is loaded with a selector. All iAPX 286 instructions which load a segment register will reference the memory based tables without additional software. The memory based tables contain 8 byte values called descriptors.

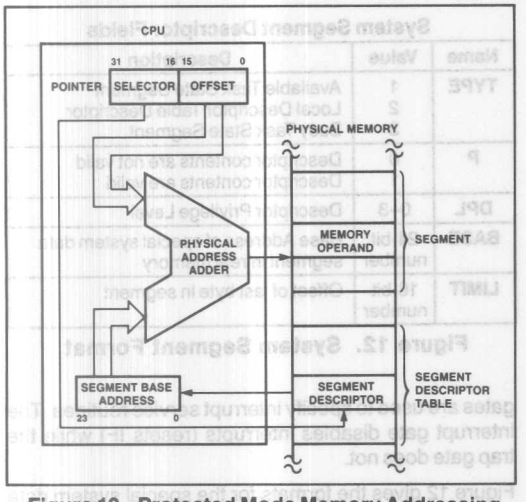


Figure 10. Protected Mode Memory Addressing

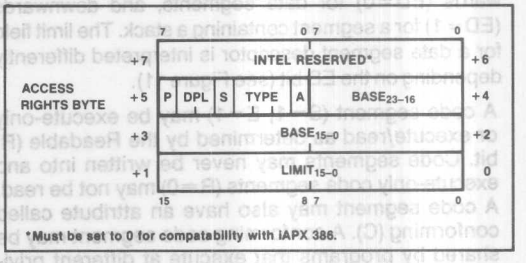
DESCRIPTORS

Descriptors define the use of memory. Special types of descriptors also define new functions for transfer of control and task switching. The 80286 has segment descriptors for code, stack and data segments, and system control descriptors for special system data segments and control transfer operations. Descriptor accesses are performed as locked bus operations to assure descriptor integrity in multi-processor systems.

CODE AND DATA SEGMENT DESCRIPTORS

Besides segment base addresses, code and data descriptors contain other segment attributes including segment size (1 to 64K bytes), access rights (read only, read/write, execute only, and execute/read), and presence in memory (for virtual memory systems) (See Figure 11). Any segment usage violating a segment attribute indicated by the segment descriptor will prevent the memory cycle and cause an exception or interrupt.

Segment Descriptor



*Must be set to 0 for compatability with iAPX 386.

Access Rights Byte Definition

	Bit Position	Name	Function	
	7	Present (P)	P = 1	Segment is mapped into physical memory.
			P = 0	No mapping to physical memory exists, base and limit are not used.
	6–5	Descriptor Privilege Level (DPL)		Segment privilege attribute used in privilege tests.
	4	Segment Descriptor (S)	S = 1	Code or Data segment descriptor
			S = 0	Non-segment descriptor
Type Field Definition	3	Executable (E)	E = 0	Data segment descriptor type is:
	2	Expansion Direction (ED)	ED = 0	Grow up segment, offsets must be ≤ limit.
			ED = 1	Grow down segment, offsets must be > limit.
	1	Writeable (W)	W = 0	Data segment may not be written into.
			W = 1	Data segment may be written into.
	3	Executable (E)	E = 1	Code Segment Descriptor type is:
	2	Conforming (C)	C = 1	Code segment may only be executed when CPL ≥ DPL.
	1	Readable (R)	R = 0	Code segment may not be read.
			R = 1	Code segment may be read.
	0	Accessed (A)	A = 0	Segment has not been accessed.
			A = 1	Segment selector has been loaded into segment register or used by selector test instructions.

Data Segment

Code Segment

Figure 11. Code and Data Segment Descriptor Formats

code segments and data segments. Both types are identified and defined by segment descriptors. Code segments are identified by the executable (E) bit set to 1 in the descriptor access rights byte. The access rights byte of both code and data segment descriptor types have three fields in common: present (P) bit, Descriptor Privilege Level (DPL), and accessed (A) bit. If P = 0, any attempted use of this segment will cause a not-present exception. DPL specifies the privilege level of the segment descriptor. DPL effects when the descriptor may be used by a task (refer to privilege discussion below). The A bit shows whether the segment has been previously accessed for usage profiling, a necessity for virtual memory systems. The CPU will always set this bit when accessing the descriptor.

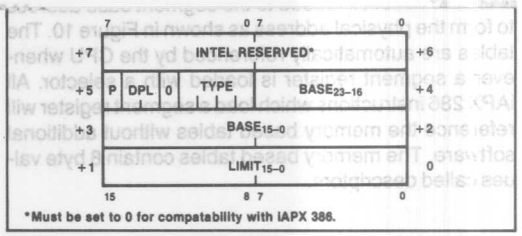
Data segments (S = 1, E = 0) may be either read-only or read-write as controlled by the W bit of the access rights byte. Read-only (W = 0) data segments may not be written into. Data segments may grow in two directions, as determined by the Expansion Direction (ED) bit: upwards (ED = 0) for data segments, and downwards (ED = 1) for a segment containing a stack. The limit field for a data segment descriptor is interpreted differently depending on the ED bit (see Figure 11).

A code segment (S = 1, E = 1) may be execute-only or execute/read as determined by the Readable (R) bit. Code segments may never be written into and execute-only code segments (R = 0) may not be read. A code segment may also have an attribute called conforming (C). A conforming code segment may be shared by programs that execute at different privilege levels. The DPL of a conforming code segment defines the range of privilege levels at which the segment may be executed (refer to privilege discussion below). The limit field identifies the last byte of a code segment.

SYSTEM CONTROL DESCRIPTORS

In addition to code and data segment descriptors, the protected mode 80286 defines system control descriptors. These descriptors define special system data segments and control transfer mechanisms in the protected environment. The special system data segment descriptors define segments which contain tables of descriptors (Local Descriptor Table Descriptor) and segments which contain the execution state of a task (Task State Segment Descriptor).

The control transfer descriptors are call gates, task gates, interrupt gates and trap gates. Gates provide a level of indirection between the source and destination of the control transfer. This indirection allows the CPU to automatically perform protection checks and control the entry point of the destination. Call gates are used to change privilege levels (see Privilege), task gates are used to perform a task switch, and interrupt and trap



System Segment Descriptor Fields

Name	Value	Description
TYPE	1 2 3	Available Task State Segment Local Descriptor Table Descriptor Busy Task State Segment
P	0 1	Descriptor contents are not valid Descriptor contents are valid
DPL	0–3	Descriptor Privilege Level
BASE	24-bit number	Base Address of special system data segment in real memory
LIMIT	16-bit number	Offset of last byte in segment

Figure 12. System Segment Format

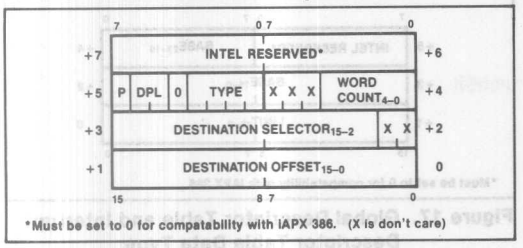
gates are used to specify interrupt service routines. The interrupt gate disables interrupts (resets IF) while the trap gate does not.

Figure 12 gives the formats for the special system data segment descriptors. The descriptors contain a 24-bit base address of the segment and a 16-bit limit. The access byte defines the type of descriptor, its state and privilege level. The descriptor contents are valid and the segment is in physical memory if P = 1. If P = 0, the segment is not valid. The DPL field is only used in Task State Segment descriptors and indicates the privilege level at which the descriptor may be used (see Privilege). Since the Local Descriptor Table descriptor may only be used by a special privileged instruction, the DPL field is not used. Bit 4 of the access byte is 0 to indicate that it is a system control descriptor. The type field specifies the descriptor type as indicated in Figure 12.

Figure 13 shows the format of the gate descriptors. The descriptor contains a destination pointer that points to the descriptor of the target segment and the entry point offset. The destination selector in an interrupt gate, trap gate, and call gate must refer to a code segment descriptor. These gate descriptors contain the entry point to prevent a program from constructing and using an illegal entry point. Task gates may only refer to a task state segment. Since task gates invoke a task switch, the destination offset is not used in the task gate.

Exception 13 is generated when the gate is used if a destination selector does not refer to the correct de-

Gate Descriptor



Gate Descriptor Fields

Name	Value	Description
TYPE	4	–Call Gate
	5	–Task Gate
	6	–Interrupt Gate
	7	–Trap Gate
P	0	–Descriptor Contents are not valid
	1	–Descriptor Contents are valid
DPL	0–3	Descriptor Privilege Level
WORD COUNT	0–31	Number of words to copy from callers stack to called procedures stack. Only used with call gate.
DESTINATION SELECTOR	16-bit selector	Selector to the target code segment (Call, Interrupt or Trap Gate) Selector to the target task state segment (Task Gate)
DESTINATION OFFSET	16-bit offset	Entry point within the target code segment

Figure 13. Gate Descriptor Format

scriptor type. The word count field is used in the call gate descriptor to indicate the number of parameters (0–31 words) to be automatically copied from the caller's stack to the stack of the called routine when a control transfer changes privilege levels. The word count field is not used by any other gate descriptor.

The access byte format is the same for all gate descriptors. P = 1 indicates that the gate contents are valid. P = 0 indicates the contents are not valid and causes exception 11 if referenced. DPL is the descriptor privilege level and specifies when this descriptor may be used by a task (refer to privilege discussion below). Bit 4 must equal 0 to indicate a system control descriptor. The type field specifies the descriptor type as indicated in Figure 13.

SEGMENT DESCRIPTOR CACHE REGISTERS

A segment descriptor cache register is assigned to each of the four segment registers (CS, SS, DS, ES). Segment descriptors are automatically loaded (cached) into a segment descriptor cache register (Figure 14) whenever the associated segment register is loaded with a selector. Only segment descriptors may be loaded into segment descriptor cache registers. Once loaded, all references to that segment of memory use the cached descriptor information instead of reaccessing memory. The descriptor cache registers are not visible to programs. No instructions exist to store their contents. They only change when a segment register is loaded.

SELECTOR FIELDS

A protected mode selector has three fields: descriptor entry index, local or global descriptor table indicator (TI), and selector privilege (RPL) as shown in Figure 15. These fields select one of two memory based tables of descriptors, select the appropriate table entry and allow high-speed testing of the selector's privilege attribute (refer to privilege discussion below).

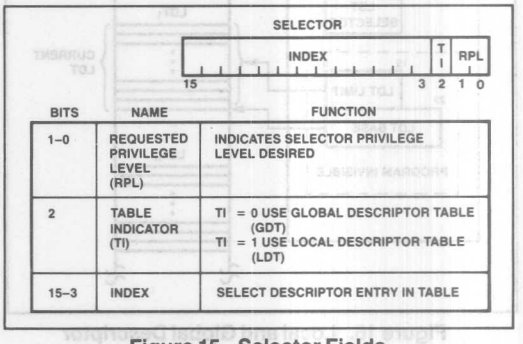


Figure 15. Selector Fields

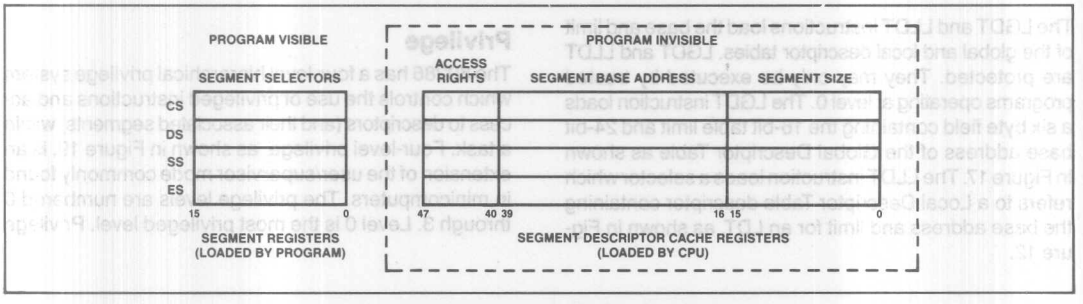


Figure 14. Descriptor Cache Registers

LOCAL AND GLOBAL DESCRIPTOR TABLES

Two tables of descriptors, called descriptor tables, contain all descriptors accessible by a task at any given time. A descriptor table is a linear array of up to 8192 descriptors. The upper 13 bits of the selector value are an index into a descriptor table. Each table has a 24-bit base register to locate the descriptor table in physical memory and a 16-bit limit register that confine descriptor access to the defined limits of the table as shown in Figure 16. A restartable exception (13) will occur if an attempt is made to reference a descriptor outside the table limits.

One table, called the Global Descriptor Table (GDT), contains descriptors available to all tasks. The other table, called the Local Descriptor Table (LDT), contains descriptors that can be private to a task. Each task may have its own private LDT. The GDT may contain all descriptor types except interrupt and trap descriptors. The LDT may contain only segment, task gate, and call gate descriptors. A segment cannot be accessed by a task if its segment descriptor does not exist in either descriptor table at the time of access.

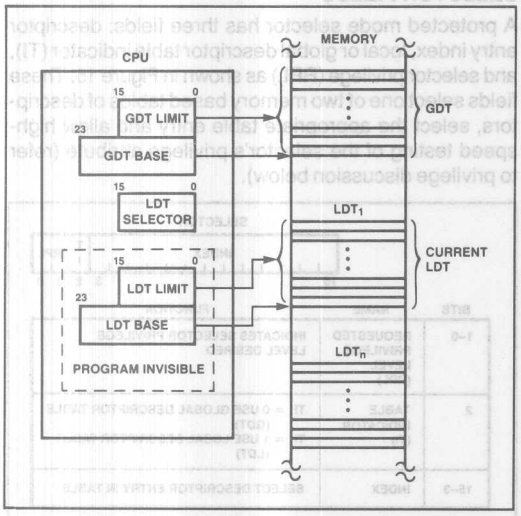


Figure 16. Local and Global Descriptor Table Definition

The LGDT and LLDT instructions load the base and limit of the global and local descriptor tables. LGDT and LLDT are protected. They may only be executed by trusted programs operating at level 0. The LGDT instruction loads a six byte field containing the 16-bit table limit and 24-bit base address of the Global Descriptor Table as shown in Figure 17. The LLDT instruction loads a selector which refers to a Local Descriptor Table descriptor containing the base address and limit for an LDT, as shown in Figure 12.

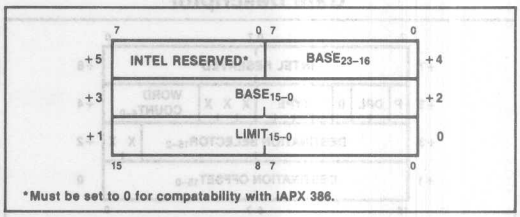


Figure 17. Global Descriptor Table and Interrupt Descriptor Table Data Type

INTERRUPT DESCRIPTOR TABLE

The protected mode 80286 has a third descriptor table, called the Interrupt Descriptor Table (IDT) (see Figure 18), used to define up to 256 interrupts. It may contain only task gates, interrupt gates and trap gates. The IDT (Interrupt Descriptor Table) has a 24-bit base and 16-bit limit register in the CPU. The protected LIDT instruction loads these registers with a six byte value of identical form to that of the LGDT instruction (see Figure 17 and Protected Mode Initialization).

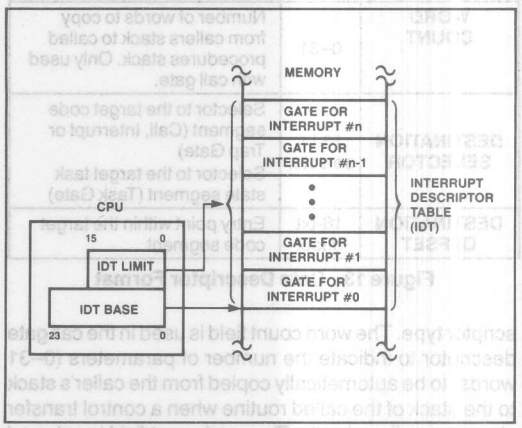


Figure 18. Interrupt Descriptor Table Definition

References to IDT entries are made via INT instructions, external interrupt vectors, or exceptions. The IDT must be at least 256 bytes in size to allocate space for all reserved interrupts.

Privilege

The 80286 has a four-level hierarchical privilege system which controls the use of privileged instructions and access to descriptors (and their associated segments) within a task. Four-level privilege, as shown in Figure 19, is an extension of the user/supervisor mode commonly found in minicomputers. The privilege levels are numbered 0 through 3. Level 0 is the most privileged level. Privilege

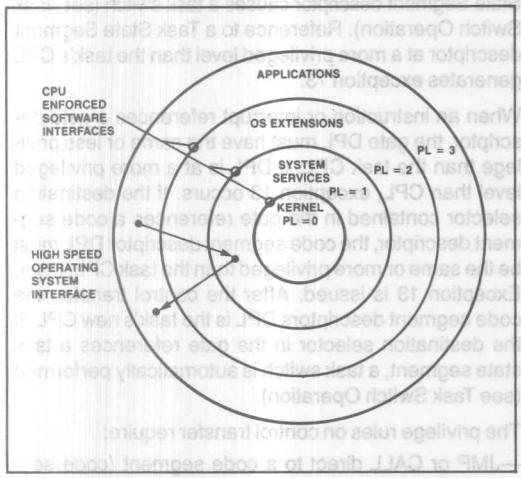


Figure 19. Hierarchical Privilege Levels

levels provide protection within a task. (Tasks are isolated by providing private LDT's for each task.) Operating system routines, interrupt handlers, and other system software can be included and protected within the virtual address space of each task using the four levels of privilege. Tasks may also have a separate stack for each privilege level.

Tasks, descriptors, and selectors have a privilege level attribute that determines whether the descriptor may be used. Task privilege effects the use of instructions and descriptors. Descriptor and selector privilege only effect access to the descriptor.

TASK PRIVILEGE

A task always executes at one of the four privilege levels. The task privilege level at any specific instant is called the Current Privilege Level (CPL) and is defined by the lower two bits of the CS register. CPL cannot change during execution in a single code segment. A task's CPL may only be changed by control transfers through gate descriptors to a new code segment (See Control Transfer). Tasks begin executing at the CPL value specified by the code segment when the task is initiated via a task switch operation. A task executing at Level 0 can access all data segments defined in the GDT and the task's LDT and is considered the most trusted level. A task executing at Level 3 has the most restricted access to data and is considered the least trusted level.

DESCRIPTOR PRIVILEGE

Descriptor privilege is specified by the Descriptor Privilege Level (DPL) field of the descriptor access byte. DPL specifies the least trusted task privilege level (CPL) at which a task may access the descriptor. Descriptors with DPL = 0 are the most protected. Only tasks executing at privilege level 0 (CPL = 0) may access them. Descriptors with DPL = 3 are the least protected (i.e. have the least restricted access) since tasks can access them when CPL = 0, 1, 2, or 3. This rule applies to all descriptors, except LDT descriptors.

SELECTOR PRIVILEGE

Selector privilege is specified by the Requested Privilege Level (RPL) field in the least significant two bits of a selector. Selector RPL may establish a less trusted privilege level than the current privilege level for the use of a selector. This level is called the task's effective privilege level (EPL). RPL can only reduce the scope of a task's access to data with this selector. A task's effective privilege is the numeric maximum of RPL and CPL. A selector with RPL = 0 imposes no additional restriction on its use while a selector with RPL = 3 can only refer to segments at privilege Level 3 regardless of the task's CPL. RPL is generally used to verify that pointer parameters passed to a more trusted procedure are not allowed to use data at a more privileged level than the caller (refer to pointer testing instructions).

Descriptor Access and Privilege Validation

Determining the ability of a task to access a segment involves the type of segment to be accessed, the instruction used, the type of descriptor used and CPL, RPL, and DPL. The two basic types of segment accesses are control transfer (selectors loaded into CS) and data (selectors loaded into DS, ES or SS).

DATA SEGMENT ACCESS

Instructions that load selectors into DS and ES must refer to a data segment descriptor or readable code segment descriptor. The CPL of the task and the RPL of the selector must be the same as or more privileged (numerically equal to or lower than) than the descriptor DPL. In general, a task can only access data segments at the same or less privileged levels than the CPL or RPL (whichever is numerically higher) to prevent a program from accessing data it cannot be trusted to use.

An exception to the rule is a readable conforming code segment. This type of code segment can be read from any privilege level.

If the privilege checks fail (e.g. DPL is numerically less than the maximum of CPL and RPL) or an incorrect type of descriptor is referenced (e.g. gate descriptor or execute only code segment) exception 13 occurs. If the segment is not present, exception 11 is generated.

descriptor privilege (DPL) and RPL must equal CPL. All other descriptor types or a privilege level violation will cause exception 13. A not present fault causes exception 12.

CONTROL TRANSFER

Four types of control transfer can occur when a selector is loaded into CS by a control transfer operation (see Table 10). Each transfer type can only occur if the operation which loaded the selector references the correct descriptor type. Any violation of these descriptor usage rules (e.g. JMP through a call gate or RET to a Task State Segment) will cause exception 13.

The ability to reference a descriptor for control transfer is also subject to rules of privilege. A CALL or JUMP instruction may only reference a code segment descriptor with DPL equal to the task CPL or a conforming segment with DPL of equal or greater privilege than CPL. The RPL of the selector used to reference the code descriptor must have as much privilege as CPL.

RET and IRET instructions may only reference code segment descriptors with descriptor privilege equal to or less privileged than the task CPL. The selector loaded into CS is the return address from the stack. After the return, the selector RPL is the task's new CPL. If CPL changes, the old stack pointer is popped after the return address.

When a JMP or CALL references a Task State Segment descriptor, the descriptor DPL must be the same or less privileged than the task's CPL. Reference to a valid Task descriptor at a more privileged level than the task's CPL generates exception 13.

When an instruction or interrupt references a gate descriptor, the gate DPL must have the same or less privilege than the task CPL. If DPL is at a more privileged level than CPL, exception 13 occurs. If the destination selector contained in the gate references a code segment descriptor, the code segment descriptor DPL must be the same or more privileged than the task CPL. If not, Exception 13 is issued. After the control transfer, the code segment descriptors DPL is the task's new CPL. If the destination selector in the gate references a task state segment, a task switch is automatically performed (see Task Switch Operation).

The privilege rules on control transfer require:

—JMP or CALL direct to a code segment (code segment descriptor) can only be to a conforming segment with DPL of equal or greater privilege than CPL or a non-conforming segment at the same privilege level.

—interrupts within the task or calls that may change privilege levels, can only transfer control through a gate at the same or a less privileged level than CPL to a code segment at the same or more privileged level than CPL.

—return instructions that don't switch tasks can only return control to a code segment at the same or less privileged level.

—task switch can be performed by a call, jump or interrupt which references either a task gate or task state segment at the same or less privileged level.

Table 10. Descriptor Types Used for Control Transfer

Control Transfer Types	Operation Types	Descriptor Referenced	Descriptor Table
Intersegment within the same privilege level	JMP, CALL, RET, IRET*	Code Segment	GDT/LDT
Intersegment to the same or higher privilege level Interrupt within task may change CPL.	CALL	Call Gate	GDT/LDT
	Interrupt Instruction, Exception, External Interrupt	Trap or Interrupt Gate	IDT
Intersegment to a lower privilege level (changes task CPL)	RET, IRET*	Code Segment	GDT/LDT
Task Switch	CALL, JMP	Task State Segment	GDT
	CALL, JMP	Task Gate	GDT/LDT
	IRET** Interrupt Instruction, Exception, External Interrupt	Task Gate	IDT

*NT (Nested Task bit of flag word) = 0
**NT (Nested Task bit of flag word) = 1

PRIVILEGE LEVEL CHANGES

Any control transfer that changes CPL within the task, causes a change of stacks as part of the operation. Initial values of SS:SP for privilege levels 0, 1, and 2 are kept in the task state segment (refer to Task Switch Operation). During a JMP or CALL control transfer, the new stack pointer is loaded into the SS and SP registers and the previous stack pointer is pushed onto the new stack.

When returning to the original privilege level, its stack is restored as part of the RET or IRET instruction operation. For subroutine calls that pass parameters on the stack and cross privilege levels, a fixed number of words, as specified in the gate, are copied from the previous stack to the current stack. The inter-segment RET instruction with a stack adjustment value will correctly restore the previous stack pointer upon return.

Protection

The 80286 includes mechanisms to protect critical instructions that affect the CPU execution state (e.g. HLT) and code or data segments from improper usage. These mechanisms are grouped under the term "protection" and have three forms:

Restricted usage of segments (e.g. no write allowed to read-only data segments). The only segments available for use are defined by descriptors in the Local Descriptor Table (LDT) and Global Descriptor Table (GDT).

Restricted access to segments via the rules of privilege and descriptor usage.

Privileged instructions or operations that may only be executed at certain privilege levels as determined by the CPL and I/O Privilege Level (IOPL). The IOPL is defined by bits 14 and 13 of the flag word.

These checks are performed for all instructions and can be split into three categories: segment load checks (Table 11), operand reference checks (Table 12), and privileged instruction checks (Table 13). Any violation of the rules shown will result in an exception. A not-present exception related to the stack segment causes exception 12.

The IRET and POPF instructions do not perform some of their defined functions if CPL is not of sufficient privilege (numerically small enough). No exceptions or other indication are given when these conditions occur.

The IF bit is not changed if CPL > IOPL.

The IOPL field of the flag word is not changed if CPL > 0.

Table 11
Segment Register Load Checks

Error Description	Exception Number
Descriptor table limit exceeded	13
Segment descriptor not-present	11 or 12
Privilege rules violated	13
Invalid descriptor/segment type segment register load: —Read only data segment load to SS —Special control descriptor load to DS, ES, SS —Execute only segment load to DS, ES, SS —Data segment load to CS —Read/Execute code segment load to SS	13

Table 12 Operand Reference Checks

Error Description	Exception Number
Write into code segment	13
Read from execute-only code segment	13
Write to read-only data segment	13
Segment limit exceeded[1]	12 or 13

Note 1: Carry out in offset calculations is ignored.

Table 13. Privileged Instruction Checks

Error Description	Exception Number
CPL ≠ 0 when executing the following instructions: LIDT, LLDT, LGDT, LTR, LMSW, CTS, HLT	13
CPL > IOPL when executing the following instructions: INS, IN, OUTS, OUT, STI, CLI, LOCK	13

EXCEPTIONS

The 80286 detects several types of exceptions and interrupts, in protected mode (see Table 14). Most are restartable after the exceptional condition is removed. Interrupt handlers for most exceptions receive an error code, pushed on the stack after the return address, that identifies the selector involved (0 if none). The return address normally points to the failing instruction, including all leading prefixes. For a processor extension segment overrun exception, the return address will not point at the ESC instruction that caused the exception; however, the processor extension registers may contain the address of the failing instruction.

Table 14. Protected Mode Exceptions

Interrupt Vector	Function	Return Address At Failing Instruction?	Always Restart-able?	Error Code on Stack?
8	Double exception detected	Yes	No	Yes
9	Processor extension segment overrun	No	No	No
10	Invalid task state segment	Yes	Yes	Yes
11	Segment not present	Yes	Yes	Yes
12	Stack segment overrun or segment not present	Yes	Yes[1]	Yes
13	General protection	Yes	No	Yes

Note 1: When a PUSHA or POPA instruction attempts to wrap around the stack segment, the machine state after the exception will not be restartable. This condition is identified by the value of the saved SP being either 0000(H), 0001(H), FFFE(H), or FFFF(H).

All these checks are performed for all instructions and can be split into three categories: segment load checks (Table 11), operand reference checks (Table 12), and privileged instruction checks (Table 13). Any violation of the rules shown will result in an exception. A not-present exception related to the stack segment causes exception 12.

Special Operations

TASK SWITCH OPERATION

The 80286 provides a built-in task switch operation which saves the entire 80286 execution state (registers, address space, and a link to the previous task), loads a new execution state, and commences execution in the new task. Like gates, the task switch operation is invoked by executing an inter-segment JMP or CALL instruction which refers to a Task State Segment (TSS) or task gate descriptor in the GDT or LDT. An INT n instruction, exception, or external interrupt may also invoke the task switch operation by selecting a task gate descriptor in the associated IDT descriptor entry.

The TSS descriptor points at a segment (see Figure 20) containing the entire 80286 execution state while a task gate descriptor contains a TSS selector. The limit field must be > 002B(H).

Each task must have a TSS associated with it. The current TSS is identified by a special register in the 80286 called the Task Register (TR). This register contains a selector referring to the task state segment descriptor that defines the current TSS. A hidden base and limit register associated with TR are loaded whenever TR is loaded with a new selector.

The IRET instruction is used to return control to the task that called the current task or was interrupted. Bit 14 in the flag register is called the Nested Task (NT) bit. It controls the function of the IRET instruction. If NT = 0, the IRET instruction performs the regular current task return; when NT = 1, IRET performs a task switch operation back to the previous task.

When a CALL or INT instruction initiates a task switch, the old and new TSS will be marked busy and the back link field of the new TSS set to the old TSS selector. The NT bit of the new task is set by CALL or INT initiated task switches. An interrupt that does not cause a task switch will clear NT. NT may also be set or cleared by POPF or IRET instructions.

The task state segment is marked busy by changing the descriptor type field from Type 1 to Type 3. Use of a selector that references a busy task state segment causes Exception 13.

PROCESSOR EXTENSION CONTEXT SWITCHING

The context of a processor extension (such as the 80287 numerics processor) is not changed by the task switch operation. A processor extension context need only be changed when a different task attempts to use the processor extension (which still contains the context of a previous task). The 80286 detects the first use of a processor extension after a task switch by causing the processor extension not present exception (7). The interrupt handler may then decide whether a context change is necessary.

Whenever the 80286 switches tasks, it sets the Task Switched (TS) bit of the MSW. TS indicates that a processor extension context may belong to a different task than the current one. The processor extension not present exception (7) will occur when attempting to execute an ESC or WAIT instruction if TS = 1 and a processor extension is present (MP = 1 in MSW).

POINTER TESTING INSTRUCTIONS

The iAPX 286 provides several instructions to speed pointer testing and consistency checks for maintaining system integrity (see Table 15). These instructions use the memory management hardware to verify that a selector value refers to an appropriate segment without risking an exception. A condition flag (ZF) indicates whether use of the selector or segment will cause an exception.

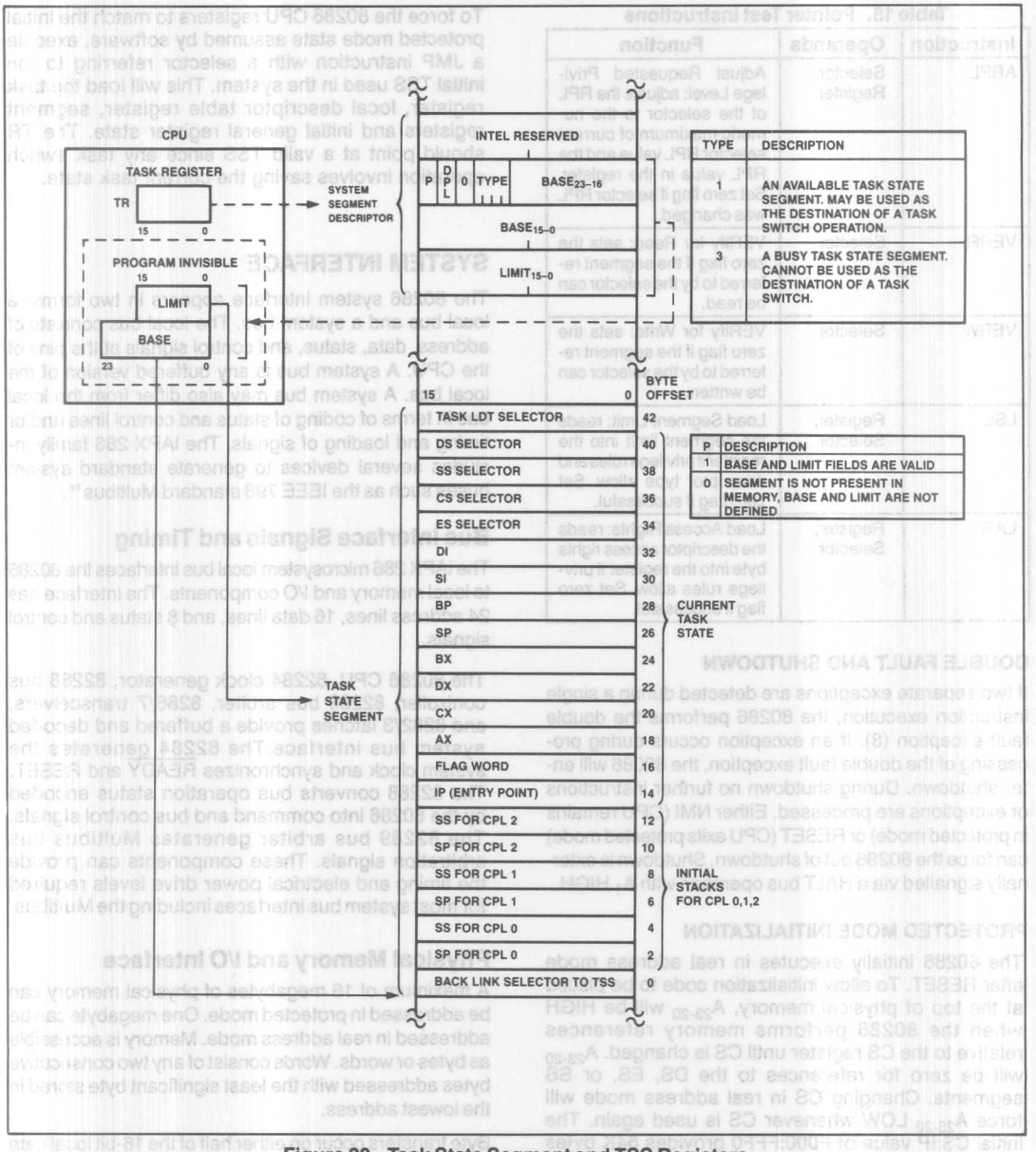


Figure 20. Task State Segment and TSS Registers

Instruction	Operands	Function
ARPL	Selector, Register	Adjust Requested Privilege Level: adjusts the RPL of the selector to the numeric maximum of current selector RPL value and the RPL value in the register. Set zero flag if selector RPL was changed.
VERR	Selector	VERify for Read: sets the zero flag if the segment referred to by the selector can be read.
VERW	Selector	VERify for Write: sets the zero flag if the segment referred to by the selector can be written.
LSL	Register, Selector	Load Segment Limit: reads the segment limit into the register if privilege rules and descriptor type allow. Set zero flag if successful.
LAR	Register, Selector	Load Access Rights: reads the descriptor access rights byte into the register if privilege rules allow. Set zero flag if successful.

DOUBLE FAULT AND SHUTDOWN

If two separate exceptions are detected during a single instruction execution, the 80286 performs the double fault exception (8). If an exception occurs during processing of the double fault exception, the 82086 will enter shutdown. During shutdown no further instructions or exceptions are processed. Either NMI (CPU remains in protected mode) or RESET (CPU exits protected mode) can force the 80286 out of shutdown. Shutdown is externally signalled via a HALT bus operation with A_1 HIGH.

PROTECTED MODE INITIALIZATION

The 80286 initially executes in real address mode after RESET. To allow initialization code to be placed at the top of physical memory, A_{23-20} will be HIGH when the 80286 performs memory references relative to the CS register until CS is changed. A_{23-20} will be zero for references to the DS, ES, or SS segments. Changing CS in real address mode will force A_{23-20} LOW whenever CS is used again. The initial CS:IP value of F000:FFF0 provides 64K bytes of code space for initialization code without changing CS.

Protected mode operation requires several registers to be initialized. The GDT and IDT base registers must refer to a valid GDT and IDT. After executing the LMSW instruction to set PE, the 80286 must immediately execute an intra-segment JMP instruction to clear the instruction queue of instructions decoded in real address mode.

protected mode state assumed by software, execute a JMP instruction with a selector referring to the initial TSS used in the system. This will load the task register, local descriptor table register, segment registers and initial general register state. The TR should point at a valid TSS since any task switch operation involves saving the current task state.

SYSTEM INTERFACE

The 80286 system interface appears in two forms: a local bus and a system bus. The local bus consists of address, data, status, and control signals at the pins of the CPU. A system bus is any buffered version of the local bus. A system bus may also differ from the local bus in terms of coding of status and control lines and/or timing and loading of signals. The iAPX 286 family includes several devices to generate standard system buses such as the IEEE 796 standard Multibus™.

Bus Interface Signals and Timing

The iAPX 286 microsystem local bus interfaces the 80286 to local memory and I/O components. The interface has 24 address lines, 16 data lines, and 8 status and control signals.

The 80286 CPU, 82284 clock generator, 82288 bus controller, 82289 bus arbiter, 8286/7 transceivers, and 8282/3 latches provide a buffered and decoded system bus interface. The 82284 generates the system clock and synchronizes $\overline{\text{READY}}$ and RESET. The 82288 converts bus operation status encoded by the 80286 into command and bus control signals. The 82289 bus arbiter generates Multibus bus arbitration signals. These components can provide the timing and electrical power drive levels required for most system bus interfaces including the Multibus.

Physical Memory and I/O Interface

A maximum of 16 megabytes of physical memory can be addressed in protected mode. One megabyte can be addressed in real address mode. Memory is accessible as bytes or words. Words consist of any two consecutive bytes addressed with the least significant byte stored in the lowest address.

Byte transfers occur on either half of the 16-bit local data bus. Even bytes are accessed over D_{7-0} while odd bytes are transferred over D_{15-8}. Even-addressed words are transferred over D_{15-0} in one bus cycle, while odd-addressed words require two bus operations. The first transfers data on D_{15-8}, and the second transfers data on D_{7-0}. Both byte data transfers occur automatically, transparent to software.

Two bus signals, A_0 and $\overline{\text{BHE}}$, control transfers over the lower and upper halves of the data bus. Even address

byte transfers are indicated by A_0 LOW and $\overline{BHE}$ HIGH. Odd address byte transfers are indicated by A_0 HIGH and $\overline{BHE}$ LOW. Both A_0 and $\overline{BHE}$ are LOW for even address word transfers.

The I/O address space contains 64K addresses in both modes. The I/O space is accessible as either bytes or words, as is memory. Byte wide peripheral devices may be attached to either the upper or lower byte of the data bus. Byte-wide I/O devices attached to the upper data byte (D_{15-8}) are accessed with odd I/O addresses. Devices on the lower data byte are accessed with even I/O addresses. An interrupt controller such as Intel's 8259A must be connected to the lower data byte (D_{7-0}) for proper return of the interrupt vector.

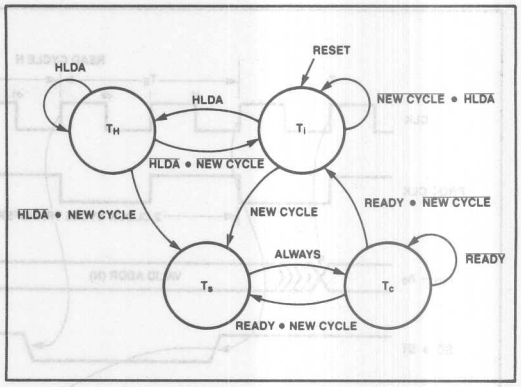


Figure 22. 80286 Bus States

Bus Operation

The 80286 uses a double frequency system clock (CLK input) to control bus timing. All signals on the local bus are measured relative to the system CLK input. The CPU divides the system clock by 2 to produce the internal processor clock, which determines bus state. Each processor clock is composed of two system clock cycles named phase 1 and phase 2. The 82284 clock generator output (PCLK) identifies the next phase of the processor clock. (See Figure 21.)

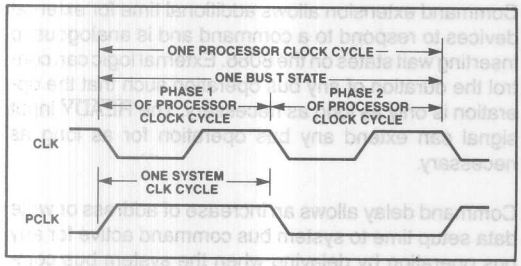


Figure 21. System and Processor Clock Relationships

Six types of bus operations are supported; memory read, memory write, I/O read, I/O write, interrupt acknowledge, and halt/shutdown. Data can be transferred at a maximum rate of one word per two processor clock cycles.

The iAPX 286 bus has three basic states: idle (T_i), send status (T_s), and perform command (T_c). The 80286 CPU also has a fourth local bus state called hold (T_h). T_h indicates that the 80286 has surrendered control of the local bus to another bus master in response to a HOLD request.

Each bus state is one processor clock long. Figure 22 shows the four 80286 local bus states and allowed transitions.

Bus States

The idle (T_i) state indicates that no data transfers are in progress or requested. The first active state (T_S) is signaled by status line $\overline{S1}$ or $\overline{S0}$ going LOW and identifying phase 1 of the processor clock. During T_S, the command encoding, the address, and data (for a write operation) are available on the 80286 output pins. The 82288 bus controller decodes the status signals and generates Multibus compatible read/write command and local transceiver control signals.

After T_s, the perform command (T_c) state is entered. Memory or I/O devices respond to the bus operation during T_c, either transferring read data to the CPU or accepting write data. T_c states may be repeated as often as necessary to assure sufficient time for the memory or I/O device to respond. The $\overline{READY}$ signal determines whether T_c is repeated.

During hold (T_h), the 80286 will float all address, data, and status output pins enabling another bus master to use the local bus. The 80286 HOLD input signal is used to place the 80286 into the T_h state. The 80286 HLDA output signal indicates that the CPU has entered T_h.

Pipelined Addressing

The 80286 uses a local bus interface with pipelined timing to allow as much time as possible for data access. Pipelined timing allows bus operations to be performed in two processor cycles, while allowing each individual bus operation to last for three processor cycles.

The timing of the address outputs is pipelined such that the address of the next bus operation becomes available during the current bus operation. Or in other words, the first clock of the next bus operation is overlapped with the last clock of the current bus operation. Therefore, address decode and routing logic can operate in ad-

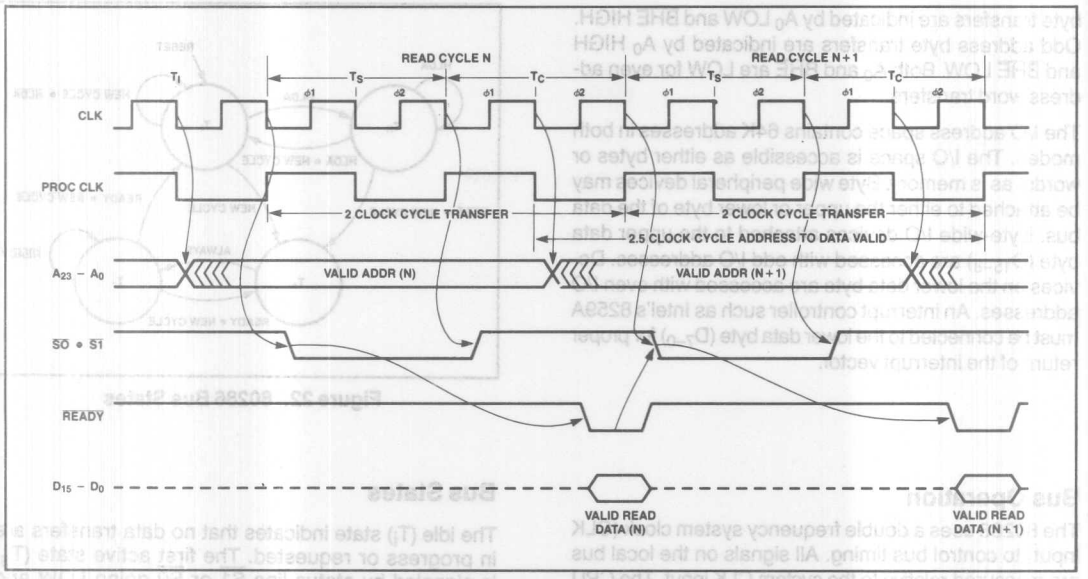


Figure 23. Basic Bus Cycle

vance of the next bus operation. External address latches may hold the address stable for the entire bus operation, and provide additional AC and DC buffering.

The 80286 does not maintain the address of the current bus operation during all T_C states. Instead, the address for the next bus operation may be emitted during phase 2 of any T_C. The address remains valid during phase 1 of the first T_C to guarantee hold time, relative to ALE, for the address latch inputs.

Bus Control Signals

The 82288 bus controller provides control signals; address latch enable (ALE), Read/Write commands, data transmit/receive (DT/$\overline{R}$), and data enable (DEN) that control the address latches, data transceivers, write enable, and output enable for memory and I/O systems.

The Address Latch Enable (ALE) output determines when the address may be latched. ALE provides at least one system CLK period of address hold time from the end of the previous bus operation until the address for the next bus operation appears at the latch outputs. This address hold time is required to support Multibus® and common memory systems.

The data bus transceivers are controlled by 82288 outputs Data Enable (DEN) and Data Transmit/Receive (DT/$\overline{R}$). DEN enables the data transceivers; while DT/$\overline{R}$ controls transceiver direction. DEN and DT/$\overline{R}$ are timed to prevent bus contention between the bus master, data bus transceivers, and system data bus tranceivers.

Command Timing Controls

Two system timing customization options, command extension and command delay, are provided on the iAPX 286 local bus.

Command extension allows additional time for external devices to respond to a command and is analogous to inserting wait states on the 8086. External logic can control the duration of any bus operation such that the operation is only as long as necessary. The $\overline{READY}$ input signal can extend any bus operation for as long as necessary.

Command delay allows an increase of address or write data setup time to system bus command active for any bus operation by delaying when the system bus command becomes active. Command delay is controlled by the 82288 CMDLY input. After T_S, the bus controller samples CMDLY at each failing edge of CLK. If CMDLY is HIGH, the 82288 will not activate the command signal. When CMDLY is LOW, the 82288 will activate the command signal. After the command becomes active, the CMDLY input is not sampled.

When a command is delayed, the available response time from command active to return read data or accept write data is less. To customize system bus timing, an address decoder can determine which bus operations require delaying the command. The CMDLY input does not affect the timing of ALE, DEN, or DT/$\overline{R}$.

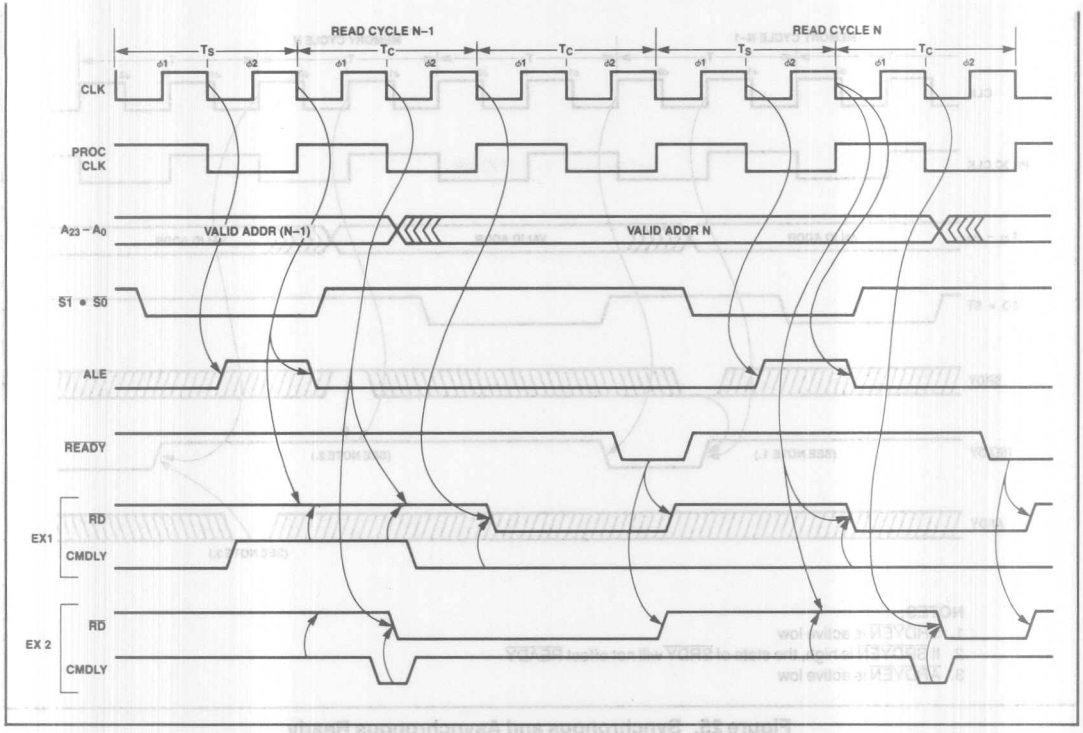


Figure 24. CMDLY Controls and Leading Edge of the Command

Figure 24 illustrates four uses of CMDLY. Example 1 shows delaying the read command two system CLKs for cycle N-1 and no delay for cycle N, and example 2 shows delaying the read command one system CLK for cycle N-1 and one system CLK delay for cycle N.

Bus Cycle Termination

At maximum transfer rates, the iAPX 286 bus alternates between the status and command states. The bus status signals become inactive after T_s so that they may correctly signal the start of the next bus operation after the completion of the current cycle. No external indication of T_c exists on the iAPX 286 local bus. The bus master and bus controller enter T_c directly after T_s and continue executing T_c cycles until terminated by $\overline{READY}$.

$\overline{READY}$ Operation

The current bus master and 82288 bus controller terminate each bus operation simultaneously to achieve maximum bus bandwidth. Both are informed in advance by $\overline{READY}$ active which identifies the last T_c cycle of the current bus operation. The bus master and bus controller must see the same sense of the $\overline{READY}$ signal, thereby requiring $\overline{READY}$ be synchronous to the system clock.

Synchronous Ready

The 82284 clock generator provides $\overline{READY}$ synchronization from both synchronous and asynchronous sources (see Figure 25). The synchronous ready input ($\overline{SRDY}$) of the clock generator is sampled with the falling edge of CLK at the end of phase 1 of each T_c. The state of $\overline{SRDY}$ is then broadcast to the bus master and bus controller via the $\overline{READY}$ output line.

Asynchronous Ready

Many systems have devices or subsystems that are asynchronous to the system clock. As a result, their ready outputs cannot be guaranteed to meet the 82284 $\overline{SRDY}$ setup and hold time requirements. The 82284 asynchronous ready input ($\overline{ARDY}$) is designed to accept such signals. The $\overline{ARDY}$ input is sampled at the beginning of each T_c cycle by 82284 synchronization logic. This provides a system CLK cycle time to resolve its value before broadcasting it to the bus master and bus controller.

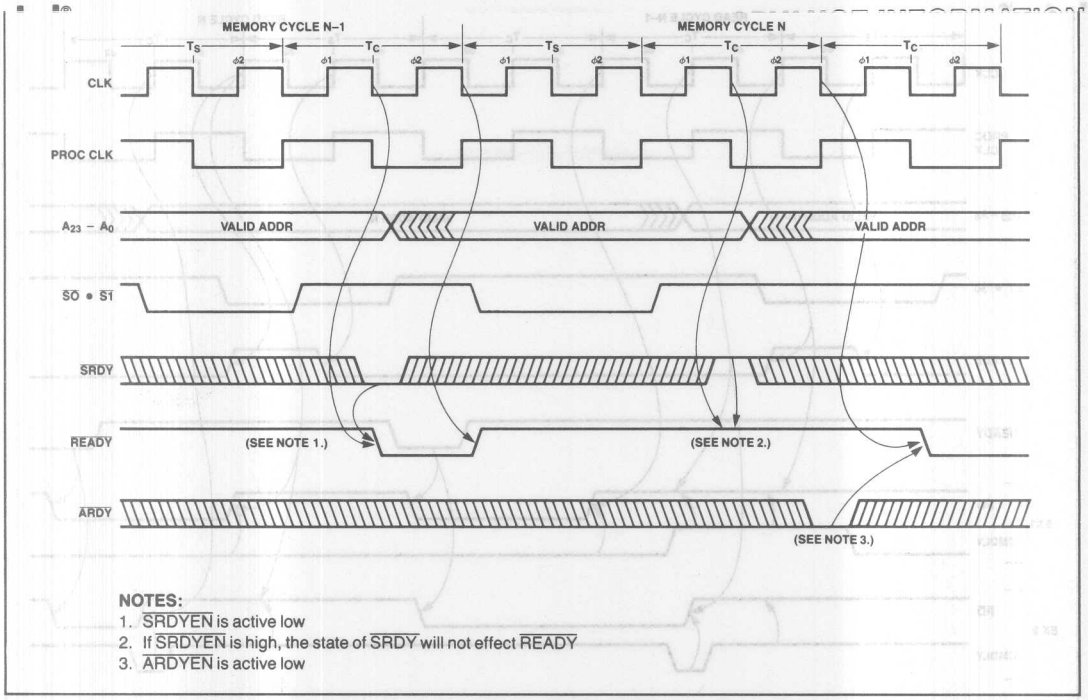


Figure 25. Synchronous and Asynchronous Ready

ARDY or ARDYEN must be HIGH at the end of T_S. ARDY cannot be used to terminate bus cycle with no wait states.

Each ready input of the 82284 has an enable pin (SRDYEN and ARDYEN) to select whether the current bus operation will be terminated by the synchronous or asynchronous ready. Either of the ready inputs may terminate a bus operation. These enable inputs are active low and have the same timing as their respective ready inputs. Address decode logic usually selects whether the current bus operation should be terminated by ARDY or SRDY.

Data Bus Control

Figures 26, 27, and 28 show how the DT/R, DEN, data bus, and address signals operate for different combinations of read, write, and idle bus operations. DT/R goes active (LOW) for a read operaton. DT/R remains HIGH before, during, and between write operations.

The data bus is driven with write data during the second phase of T_s. The delay in write data timing allows the read data drivers, from a previous read cycle, sufficient time to enter 3-state OFF before the 80286 CPU begins driving the local data bus for write operations. Write data will always remain valid for one system clock past the last T_c to provide sufficient hold time for Multibus or other similar memory or I/O systems. During write-read or write-idle sequences the data bus enters 3-state OFF during the second phase of the processor cycle after the last T_c. In a write-write sequence the data bus does not enter 3-state OFF between T_c and T_s.

Bus Usage

The 80286 local bus may be used for several functions: instruction data transfers, data transfers by other bus masters, instruction fetching, processor extension data transfers, interrupt acknowledge, and halt/shutdown. This section describes local bus activities which have special signals or requirements.

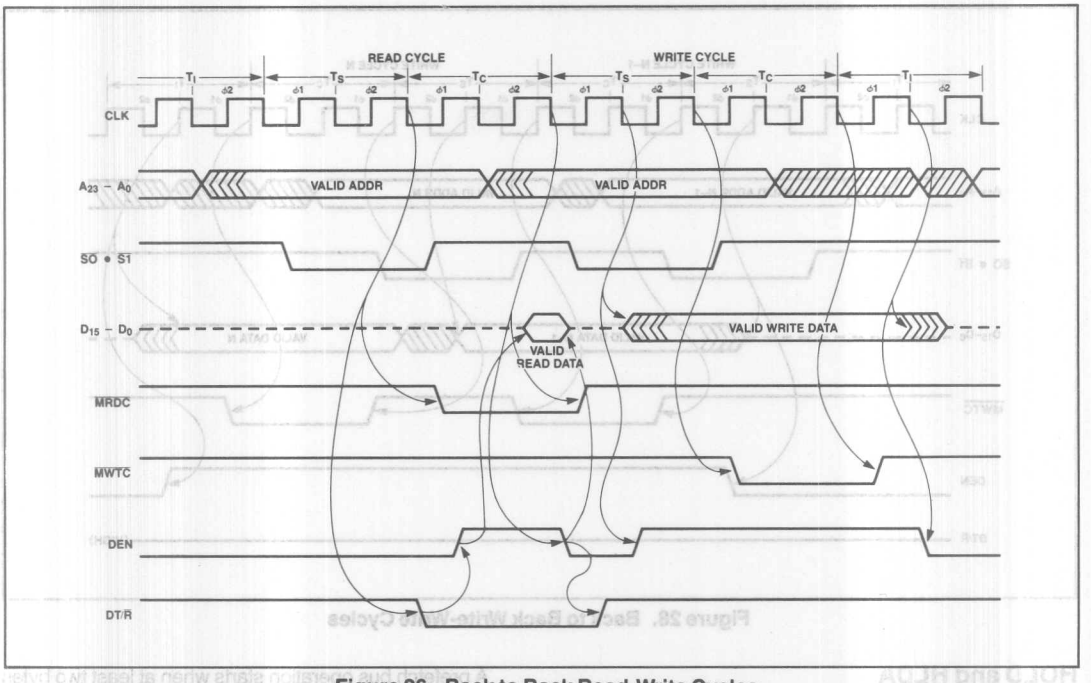


Figure 26. Back to Back Read-Write Cycles

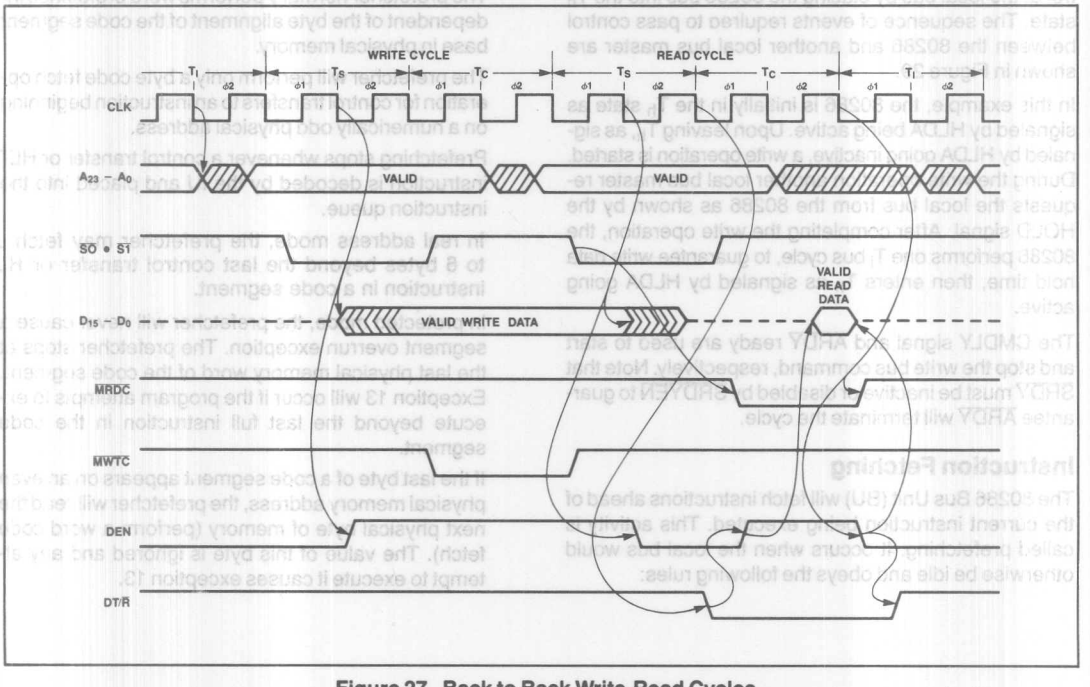


Figure 27. Back to Back Write-Read Cycles

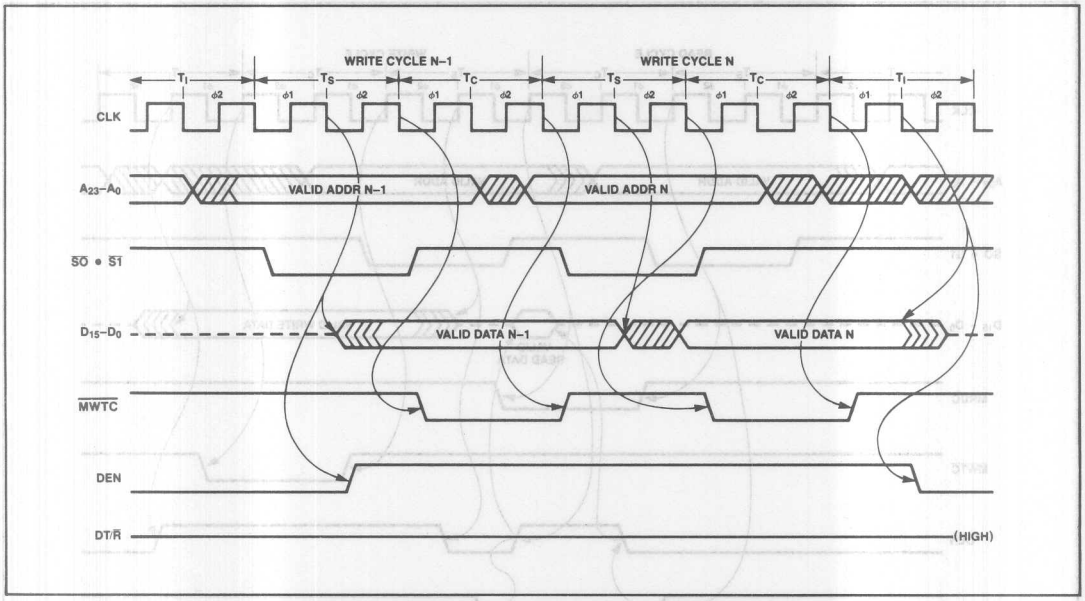


Figure 28. Back to Back Write-Write Cycles

HOLD and HLDA

HOLD and HLDA allow another bus master to gain control of the local bus by placing the 80286 bus into the T_h state. The sequence of events required to pass control between the 80286 and another local bus master are shown in Figure 29.

In this example, the 80286 is initially in the T_h state as signaled by HLDA being active. Upon leaving T_h, as signaled by HLDA going inactive, a write operation is started. During the write operation another local bus master requests the local bus from the 80286 as shown by the HOLD signal. After completing the write operation, the 80286 performs one T_i bus cycle, to guarantee write data hold time, then enters T_h as signaled by HLDA going active.

The CMDLY signal and $\overline{\text{ARDY}}$ ready are used to start and stop the write bus command, respectively. Note that $\overline{\text{SRDY}}$ must be inactive or disabled by $\overline{\text{SRDYEN}}$ to guarantee $\overline{\text{ARDY}}$ will terminate the cycle.

Instruction Fetching

The 80286 Bus Unit (BU) will fetch instructions ahead of the current instruction being executed. This activity is called prefetching. It occurs when the local bus would otherwise be idle and obeys the following rules:

A prefetch bus operation starts when at least two bytes of the 6-byte prefetch queue are empty.

The prefetcher normally performs word prefetches independent of the byte alignment of the code segment base in physical memory.

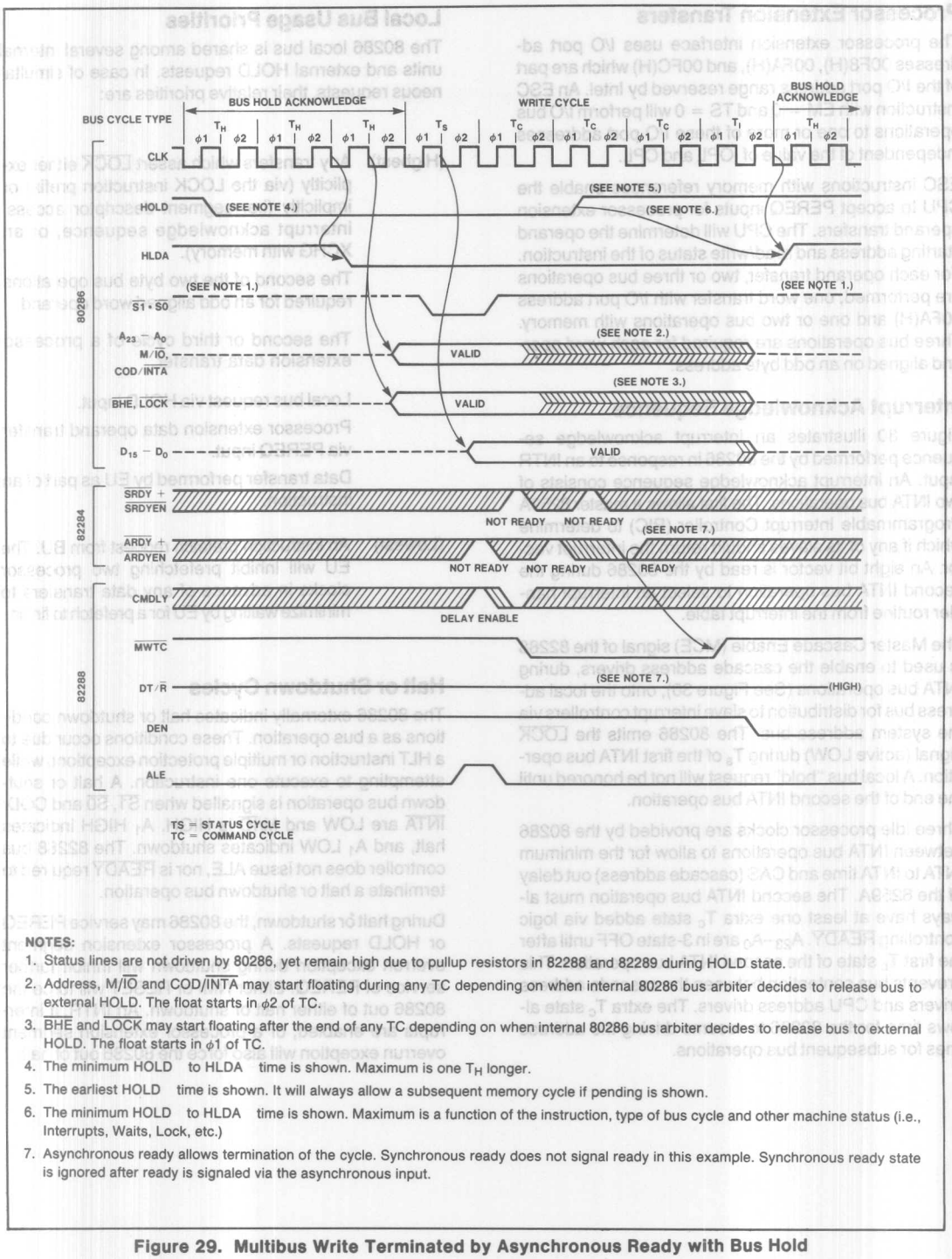
The prefetcher will perform only a byte code fetch operation for control transfers to an instruction beginning on a numerically odd physical address.

Prefetching stops whenever a control transfer or HLT instruction is decoded by the IU and placed into the instruction queue.

In real address mode, the prefetcher may fetch up to 6 bytes beyond the last control transfer or HLT instruction in a code segment.

In protected mode, the prefetcher will never cause a segment overrun exception. The prefetcher stops at the last physical memory word of the code segment. Exception 13 will occur if the program attempts to execute beyond the last full instruction in the code segment.

If the last byte of a code segment appears on an even physical memory address, the prefetcher will read the next physical byte of memory (perform a word code fetch). The value of this byte is ignored and any attempt to execute it causes exception 13.



NOTES:

1. Status lines are not driven by 80286, yet remain high due to pullup resistors in 82288 and 82289 during HOLD state.
2. Address, M/$\overline{IO}$ and COD/$\overline{INTA}$ may start floating during any TC depending on when internal 80286 bus arbiter decides to release bus to external HOLD. The float starts in $\phi 2$ of TC.
3. $\overline{BHE}$ and $\overline{LOCK}$ may start floating after the end of any TC depending on when internal 80286 bus arbiter decides to release bus to external HOLD. The float starts in $\phi 1$ of TC.
4. The minimum HOLD to HLDA time is shown. Maximum is one T_H longer.
5. The earliest HOLD time is shown. It will always allow a subsequent memory cycle if pending is shown.
6. The minimum HOLD to HLDA time is shown. Maximum is a function of the instruction, type of bus cycle and other machine status (i.e., Interrupts, Waits, Lock, etc.)
7. Asynchronous ready allows termination of the cycle. Synchronous ready does not signal ready in this example. Synchronous ready state is ignored after ready is signaled via the asynchronous input.

Figure 29. Multibus Write Terminated by Asynchronous Ready with Bus Hold

The processor extension interface uses I/O port addresses 00F8(H), 00FA(H), and 00FC(H) which are part of the I/O port address range reserved by Intel. An ESC instruction with EM = 0 and TS = 0 will perform I/O bus operations to one or more of these I/O port addresses independent of the value of IOPL and CPL.

ESC instructions with memory references enable the CPU to accept PEREQ inputs for processor extension operand transfers. The CPU will determine the operand starting address and read/write status of the instruction. For each operand transfer, two or three bus operations are performed, one word transfer with I/O port address 00FA(H) and one or two bus operations with memory. Three bus operations are required for each word operand aligned on an odd byte address.

Interrupt Acknowledge Sequence

Figure 30 illustrates an interrupt acknowledge sequence performed by the 80286 in response to an INTR input. An interrupt acknowledge sequence consists of two INTA bus operations. The first allows a master 8259A Programmable Interrupt Controller (PIC) to determine which if any of its slaves should return the interrupt vector. An eight bit vector is read by the 80286 during the second INTA bus operation to select an interrupt handler routine from the interrupt table.

The Master Cascade Enable (MCE) signal of the 82288 is used to enable the cascade address drivers, during INTA bus operations (See Figure 30), onto the local address bus for distribution to slave interrupt controllers via the system address bus. The 80286 emits the $\overline{LOCK}$ signal (active LOW) during T_s of the first INTA bus operation. A local bus "hold" request will not be honored until the end of the second INTA bus operation.

Three idle processor clocks are provided by the 80286 between INTA bus operations to allow for the minimum INTA to INTA time and CAS (cascade address) out delay of the 8259A. The second INTA bus operation must always have at least one extra T_c state added via logic controlling $\overline{READY}$. A_{23}–A_0 are in 3-state OFF until after the first T_c state of the second INTA bus operation. This prevents bus contention between the cascade address drivers and CPU address drivers. The extra T_c state allows time for the 80286 to resume driving the address lines for subsequent bus operations.

The 80286 local bus is shared among several internal units and external HOLD requests. In case of simultaneous requests, their relative priorities are:

(Highest) Any transfers which assert $\overline{LOCK}$ either explicitly (via the LOCK instruction prefix) or implicitly (i.e. segment descriptor access, interrupt acknowledge sequence, or an XCHG with memory).

The second of the two byte bus operations required for an odd aligned word operand.

The second or third cycle of a processor extension data transfer.

Local bus request via HOLD input.

Processor extension data operand transfer via PEREQ input.

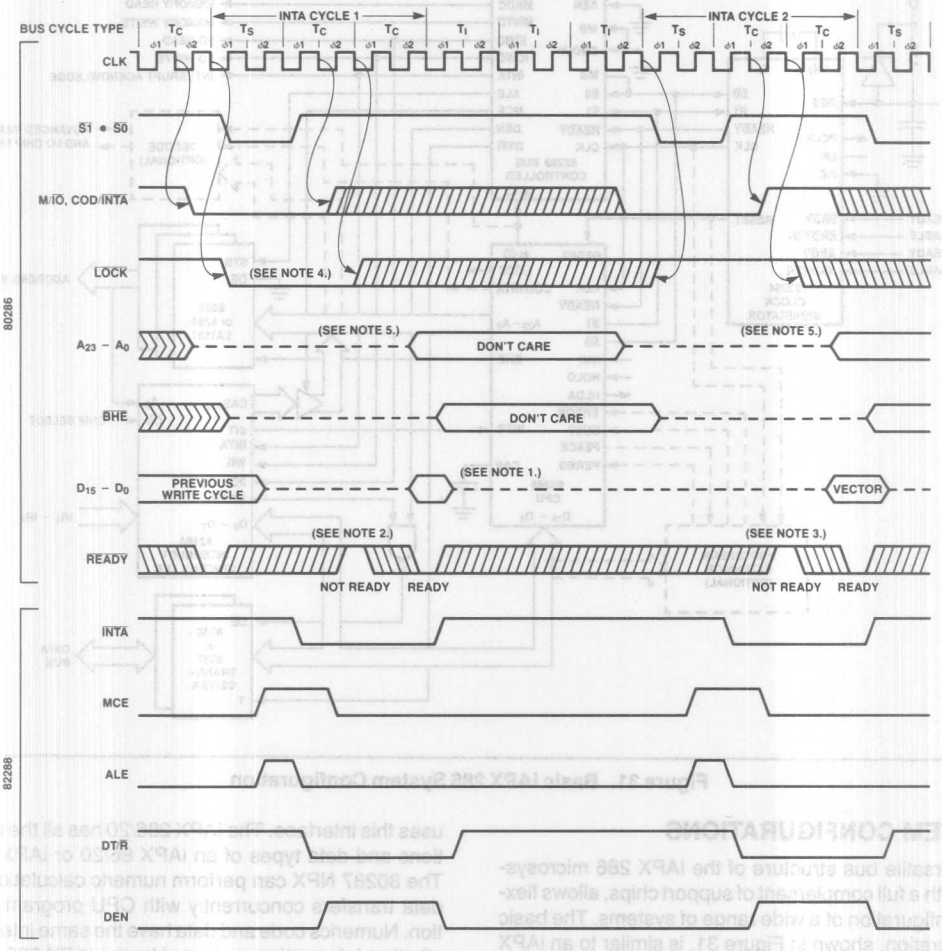
Data transfer performed by EU as part of an instruction.

(Lowest) An instruction prefetch request from BU. The EU will inhibit prefetching two processor clocks in advance of any data transfers to minimize waiting by EU for a prefetch to finish.

Halt or Shutdown Cycles

The 80286 externally indicates halt or shutdown conditions as a bus operation. These conditions occur due to a HLT instruction or multiple protection exceptions while attempting to execute one instruction. A halt or shutdown bus operation is signalled when $\overline{S1}$, $\overline{S0}$ and COD/$\overline{INTA}$ are LOW and M/$\overline{IO}$ is HIGH. A_1 HIGH indicates halt, and A_1 LOW indicates shutdown. The 82288 bus controller does not issue ALE, nor is $\overline{READY}$ required to terminate a halt or shutdown bus operation.

During halt or shutdown, the 80286 may service PEREQ or HOLD requests. A processor extension segment overrun exception during shutdown will inhibit further service of PEREQ. Either NMI or RESET will force the 80286 out of either halt or shutdown. An INTR, if interrupts are enabled, or a processor extension segment overrun exception will also force the 80286 out of halt.



NOTES:

1. Data is ignored.
2. First INTA cycle should have at least one wait state inserted to meet 8259A minimum INTA pulse width.
3. Second INTA cycle must have at least one wait state inserted since the CPU will not drive A_{23} – A_0, $\overline{BHE}$, and $\overline{LOCK}$ until after the first TC state.

 The CPU imposed one/clock delay prevents bus contention between cascade address buffer being disabled by MCE ↑ and address outputs.

 Without the wait state, the 80286 address will not be valid for a memory cycle started immediately after the second INTA cycle. The 8259A also requires one wait state for minimum INTA pulse width.
4. $\overline{LOCK}$ is active for the first INTA cycle to prevent the 82289 from releasing the bus between INTA cycles in a multi-master system.
5. A_{23} – A_0 exits 3-state OFF during $\phi 2$ of the second T_C in the INTA cycle.

Figure 30. Interrupt Acknowledge Sequence

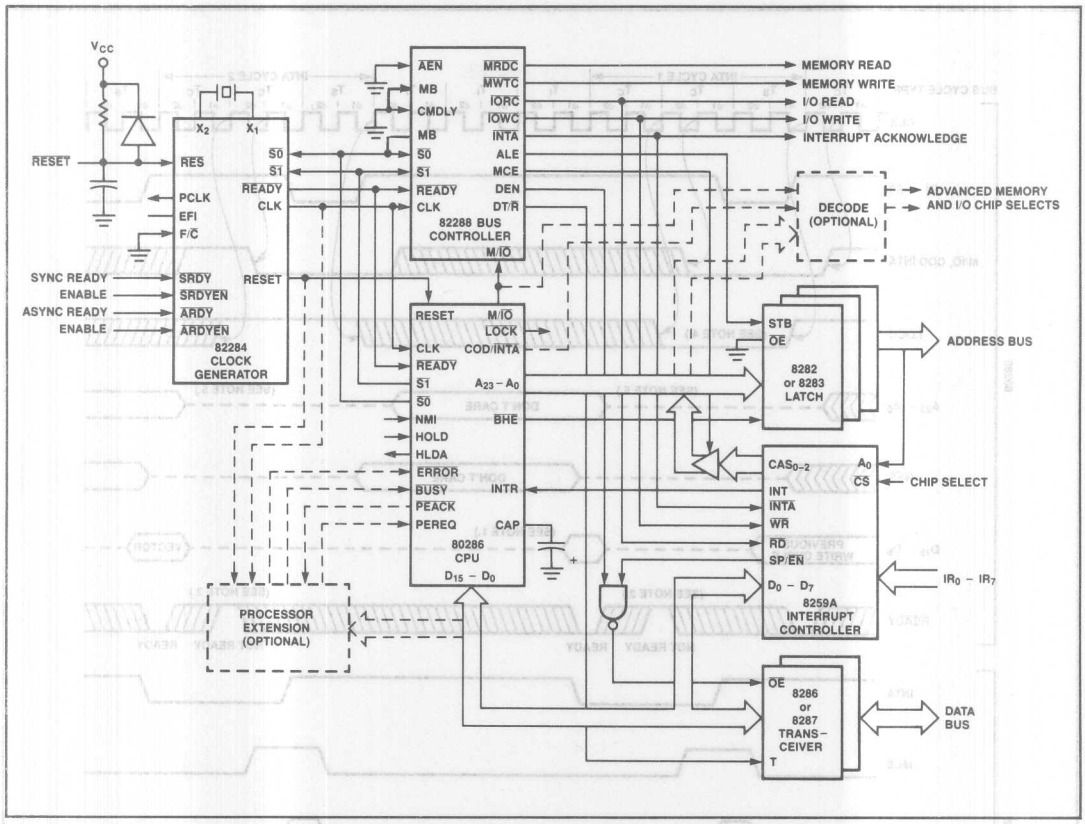


Figure 31. Basic iAPX 286 System Configuration

SYSTEM CONFIGURATIONS

The versatile bus structure of the iAPX 286 microsystem, with a full complement of support chips, allows flexible configuration of a wide range of systems. The basic configuration, shown in Figure 31, is similar to an iAPX 86 maximum mode system. It includes the CPU plus an 8259A interrupt controller, 82284 clock generator, and the 82288 Bus Controller. The iAPX 86 latches (8282 and 8283) and transceivers (8286 and 8287) may be used in an iAPX 286 microsystem.

As indicated by the dashed lines in Figure 31, the ability to add processor extensions is an integral feature of iAPX 286 microsystems. The processor extension interface allows external hardware to perform special functions and transfer data concurrent with CPU execution of other instructions. Full system integrity is maintained because the 80286 supervises all data transfers and instruction execution for the processor extension.

The iAPX 286/20 numeric data processor which includes the 80287 numeric processor extension (NPX) uses this interface. The iAPX 286/20 has all the instructions and data types of an iAPX 86/20 or iAPX 88/20. The 80287 NPX can perform numeric calculations and data transfers concurrently with CPU program execution. Numerics code and data have the same integrity as all other information protected by the iAPX 286 protection mechanism.

The 80286 can overlap chip select decoding and address propagation during the data transfer for the previous bus operation. This information is latched into the 8282/3's by ALE during the middle of a T_S cycle. The latched chip select and address information remains stable during the bus operation while the next cycles address is being decoded and propagated into the system. Decode logic can be implemented with a high speed bipolar PROM.

The optional decode logic shown in Figure 31 takes advantage of the overlap between address and data of the 80286 bus cycle to generate advanced memory and IO-select signals. This minimizes system performance

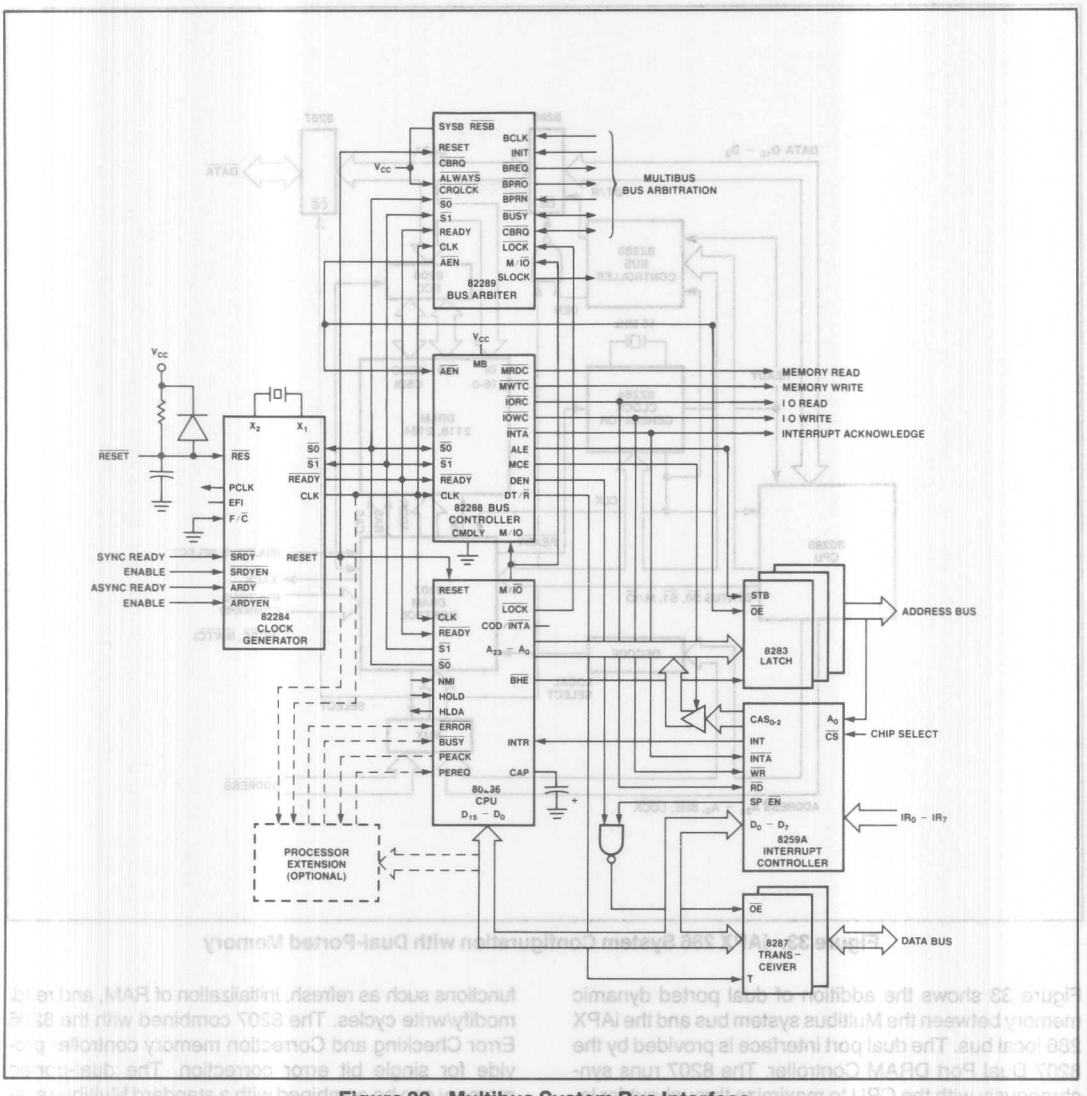


Figure 32. Multibus System Bus Interface

degradation caused by address propogation and decode delays. In addition to selecting memory and I/O, the advanced selects may be used with configurations supporting local and system buses to enable the appropriate bus interface for each bus cycle. The COD/$\overline{INTA}$ and M/$\overline{IO}$ signals are applied to the decode logic to distinguish between interrupt, I/O, code and data bus cycles.

By adding the 82289 bus arbiter chip the 80286 provides a Multibus system bus interface as shown in Figure 32. The ALE output of the 82288 for the Multibus bus is connected to its CMDLY input to delay the start of commands one system CLK as required to meet Multibus address and write data setup times. This arrangement will add at least one extra T_c state to each bus operation which uses the Multibus.

A second 82288 bus controller and additional latches and transceivers could be added to the local bus of Figure 32. This configuration allows the 80286 to support an on-board bus for local memory and peripherals, and the Multibus for system bus interfacing.

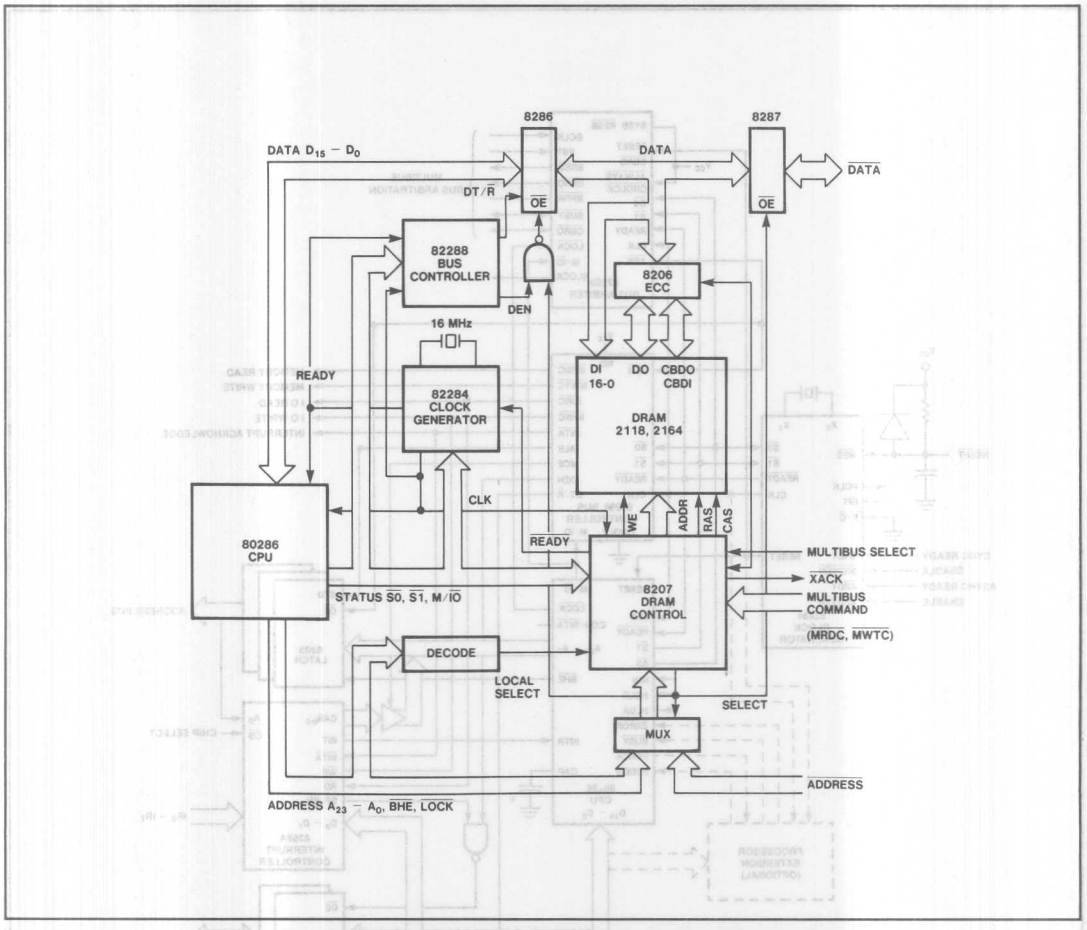


Figure 33. iAPX 286 System Configuration with Dual-Ported Memory

Figure 33 shows the addition of dual ported dynamic memory between the Multibus system bus and the iAPX 286 local bus. The dual port interface is provided by the 8207 Dual Port DRAM Controller. The 8207 runs synchronously with the CPU to maximize throughput for local memory references. It also arbitrates between requests from the local and system buses and performs functions such as refresh, initialization of RAM, and read/modify/write cycles. The 8207 combined with the 8206 Error Checking and Correction memory controller provide for single bit error correction. The dual-ported memory can be combined with a standard Multibus system bus interface to maximize performance and protection in multiprocessor system configurations.

PACKAGE

The 80286 is packaged in a 68-pin, leadless JEDEC type A hermetic leadless chip carrier. Figure 34 illustrates the package, and Figure 2 shows the pinout.

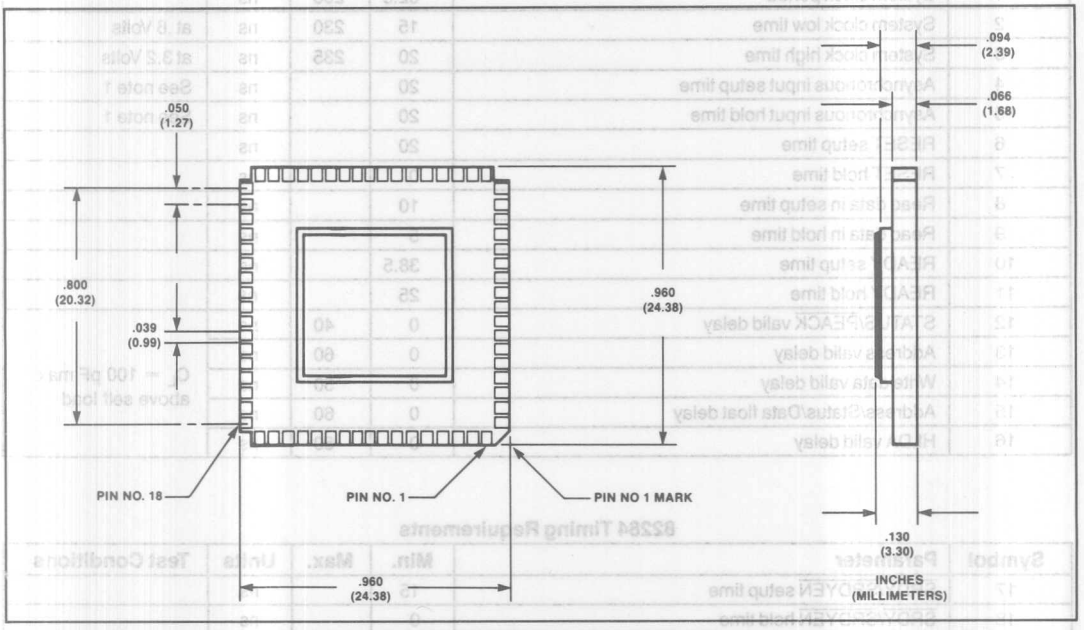


Figure 34. JEDEC Type A Package

ABSOLUTE MAXIMUM RATINGS*

Ambient Temperature Under Bias 0°C to 70°C
Storage Temperature −65°C to +150°C
Voltage on Any Pin with Respect to Ground −1.0 to +7V
Power Dissipation 3.6 Watt

**NOTICE: Stresses above those listed under "Absolute Maximum Ratings" may cause permanent damage to the device. This is a stress rating only and functional operation of the device at these or any other conditions above those indicated in the operational sections of this specification is not implied. Exposure to absolute maximum rating conditions for extended periods may affect device reliability.*

D.C. CHARACTERISTICS (80286: T_A = 0°C to 70°C, V_{CC} = 5V ± 10%)

Symbol	Parameter	Min.	Max.	Units	Test Conditions
V_{IL}	Input Low Voltage	−0.5	+0.8	V	
V_{IH}	Input High Voltage	2.0	V_{CC} + 0.5	V	
V_{OL}	Output Low Voltage		0.45	V	I_{OL} = 3.0 mA
V_{OH}	Output High Voltage	2.4		V	I_{OH} = −400 µA
I_{CC}	Power Supply Current		600	mA	T_A = 25°C
I_{LI}	Input Leakage Current		±10	µA	$0V \leq V_{IN} \leq V_{CC}$
I_{LO}	Output Leakage Current		±10	µA	$0.45V \leq V_{OUT} \leq V_{CC}$
V_{CL}	Clock Input High Voltage	−0.5	+0.6	V	
V_{CH}	Clock Input HIgh Voltage	3.8	V_{CC} + 1.0	V	
C_{IN}	Capacitance of Inputs (All input except CLK)		10	pF	fc = 1 MHz
C_O	Capacitance of I/O or outputs		20	pF	fc = 1 MHz
C_{CLK}	Capacitance of CLK Input		12	pF	f_c = 1 MHz

80286 Timing Requirements

Symbol	Parameter	Min.	Max.	Units	Test Conditions
1	System clock period	62.5	250	ns	
2	System clock low time	15	230	ns	at .6 Volts
3	System clock high time	20	235	ns	at 3.2 Volts
4	Asynchronous input setup time	20		ns	See note 1
5	Asynchronous input hold time	20		ns	See note 1
6	RESET setup time	20		ns	
7	RESET hold time	0		ns	
8	Read data in setup time	10		ns	
9	Read data in hold time	5		ns	
10	READY setup time	38.5		ns	
11	READY hold time	25		ns	
12	STATUS/$\overline{\text{PEACK}}$ valid delay	0	40	ns	C_L = 100 pF max above self load
13	Address valid delay	0	60	ns	
14	Write data valid delay	0	50	ns	
15	Address/Status/Data float delay	0	60	ns	
16	HLDA valid delay	0	60	ns	

82284 Timing Requirements

Symbol	Parameter	Min.	Max.	Units	Test Conditions
17	$\overline{\text{SRDY}}$/$\overline{\text{SRDYEN}}$ setup time	15		ns	
18	$\overline{\text{SRDY}}$/$\overline{\text{SRDYEN}}$ hold time	0		ns	
19	$\overline{\text{ARDY}}$/$\overline{\text{ARDYEN}}$ setup time	0		ns	See note 1
20	$\overline{\text{ARDY}}$/$\overline{\text{ARDYEN}}$ hold time	16		ns	See note 1.
21	PCLK delay	0	40	ns	C_L = 50 pF I_{OL} = 5 ma I_{OH} = −1 ma

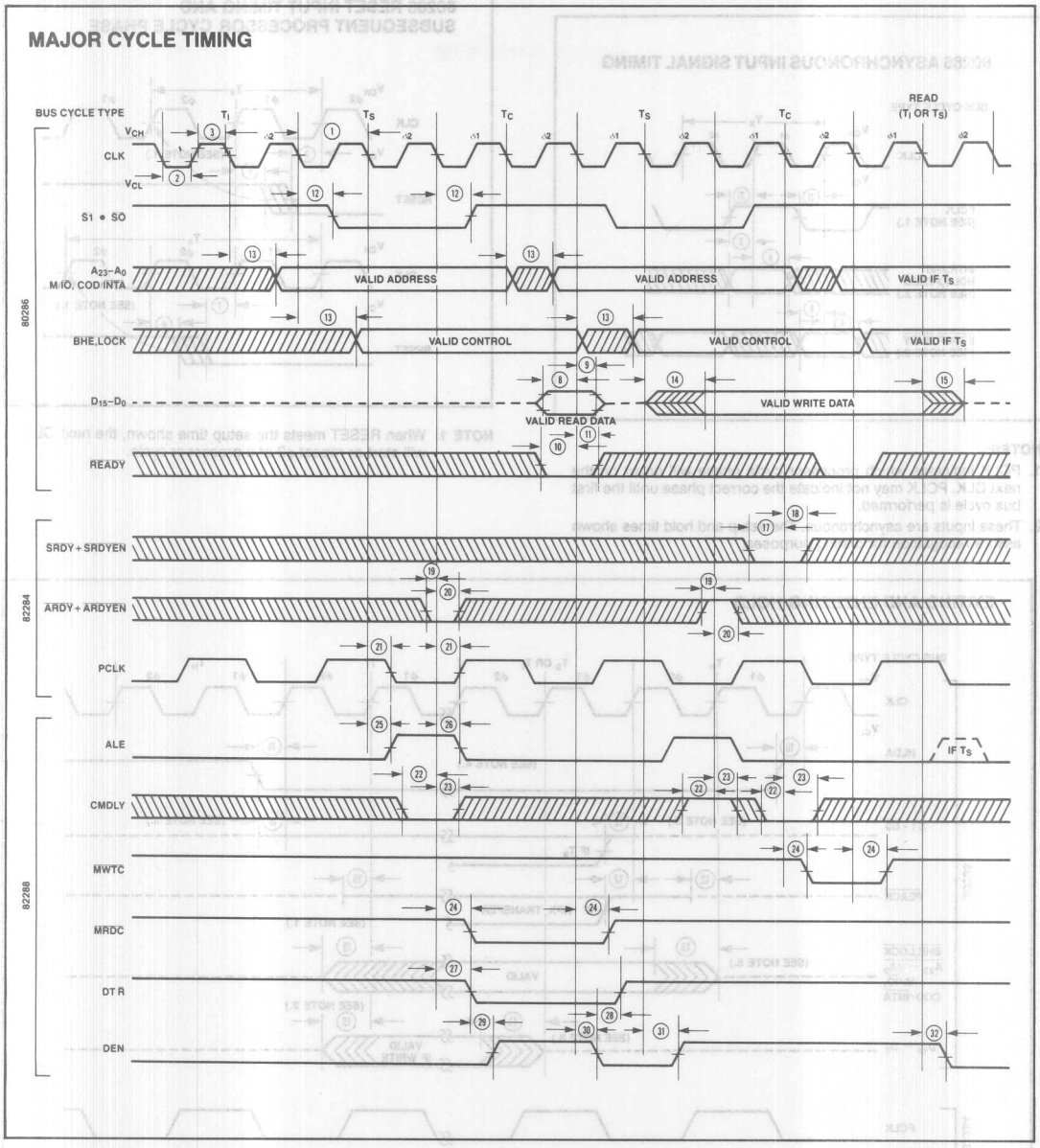
NOTE 1: These times are given for testing purposes to assure a predetermined action.

82288 Timing Requirements

Symbol	Parameter	Min.	Max.	Units	Test Conditions
22	CMDLY setup time	20		ns	
23	CMDLY hold time	0		ns	
24	Command delay	3	15	ns	C_L = 300 pF max I_{OL} = 32 ma max I_{OH} = −5 ma max
25	ALE active delay	3	15	ns	C_L = 80 pF max I_{OL} = 16 ma max I_{OH} = −1 ma max
26	ALE inactive delay	0	20	ns	
27	DT/$\overline{\text{R}}$ read active delay	0	20	ns	
28	DT/$\overline{\text{R}}$ read inactive delay	10	40	ns	
29	DEN read active delay	10	50	ns	
30	DEN read inactive delay	3	15	ns	
31	DEN write active delay	0	30	ns	
32	DEN write inactive delay	3	30	ns	

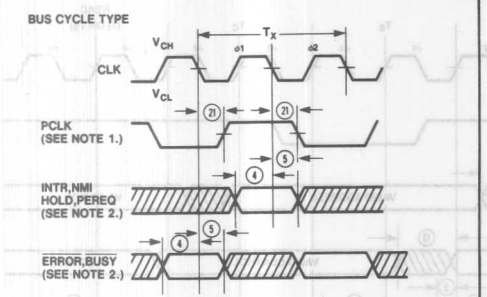
WAVEFORMS

MAJOR CYCLE TIMING

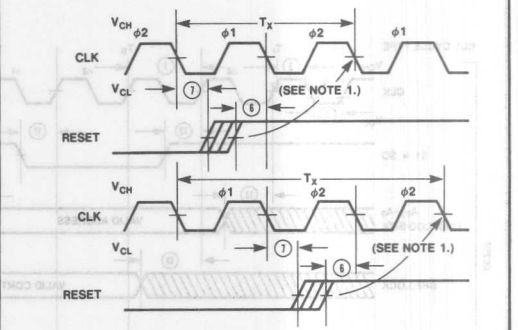


WAVEFORMS (Continued)

80286 ASYNCHRONOUS INPUT SIGNAL TIMING



80286 RESET INPUT TIMING AND SUBSEQUENT PROCESSOR CYCLE PHASE

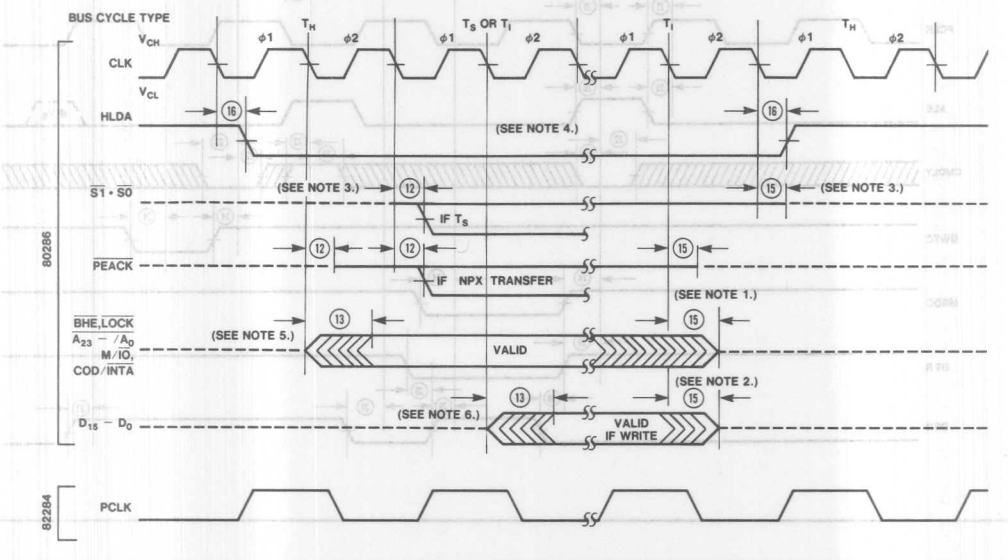


NOTE 1: When RESET meets the setup time shown, the next CLK will start or repeat $\phi 2$ of a processor cycle.

NOTES:

1. PCLK indicates which processor cycle phase will occur on the next CLK. PCLK may not indicate the correct phase until the first bus cycle is performed.
2. These inputs are asynchronous. The setup and hold times shown assure recognition for testing purposes.

EXITING AND ENTERING HOLD

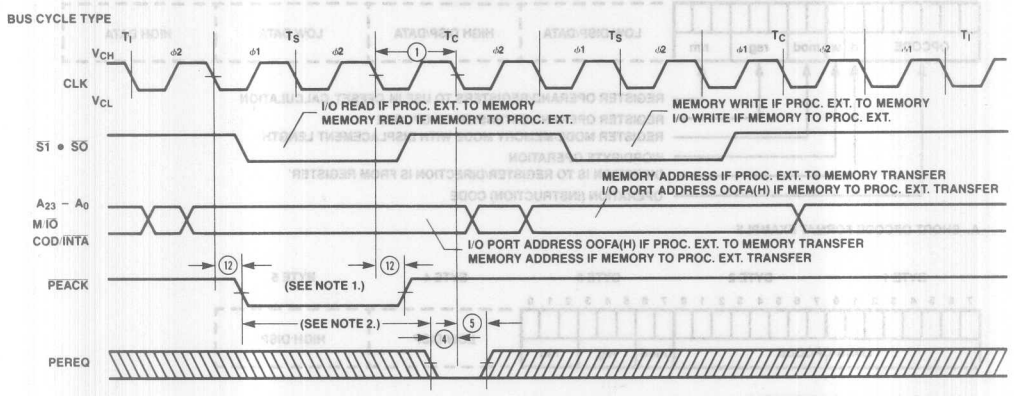


NOTES:

1. These signals may not be driven by the 80286 during the time shown. The worst case in terms of latest float time is shown.
2. The data bus will be driven as shown if the last cycle before T_i in the diagram was a write T_C.
3. The 80286 floats its status pins during T_H. External pullup resistors (in 82288) keep these signals high.
4. For HOLD request set up to HLDA, refer to Figure 29.
5. $\overline{BHE}$ and $\overline{LOCK}$ are driven at this time but will not become valid until T_S.
6. The data bus will remain in 3-state OFF if a read cycle is performed.

WAVEFORMS (Continued)

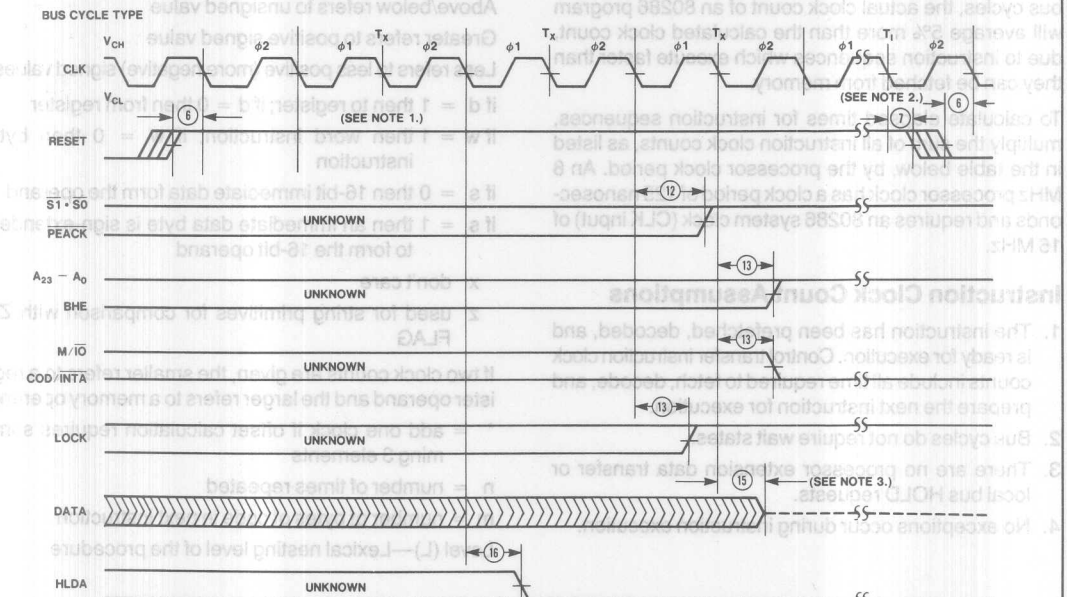
80286 PEREQ/PEACK TIMING FOR ONE TRANSFER ONLY



NOTES:

1. PEACK always goes active during the first bus operation of a processor extension data operand transfer sequence. The first bus operation will be either a memory read at operand address or I/O read at port address OOFA(H).
2. To prevent a second processor extension data operand transfer, the worst case maximum time (Shown above) is: $3X(1)-(11)_{max.}-(4)_{min.}$. The actual, configuration dependent, maximum time is: $3X(1)-(11)_{max.}-(4)_{min.}+AX2X(1)$. A is the number of extra T_C states added to either the first or second bus operation of the processor extension data operand transfer sequence.

INITIAL 80286 PIN STATE DURING RESET



NOTES:

1. Setup time for RESET ↑ may be violated with the consideration that $\phi 1$ of the processor clock may begin one system CLK period later.
2. Setup and hold times for RESET ↓ must be met for proper operation.
3. The data bus is only guaranteed to be in 3-state OFF at the time shown.

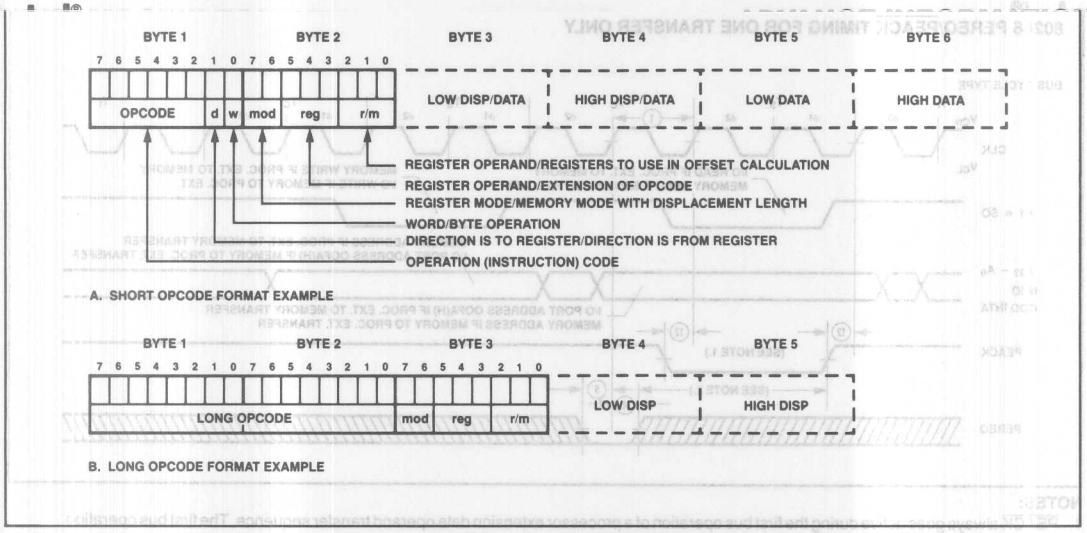


Figure 35. 80286 Instruction Format Examples

80286 INSTRUCTION SET SUMMARY

Instruction Timing Notes

The instruction clock counts listed below establish the maximum execution rate of the 80286. With no delays in bus cycles, the actual clock count of an 80286 program will average 5% more than the calculated clock count, due to instruction sequences which execute faster than they can be fetched from memory.

To calculate elapsed times for instruction sequences, multiply the sum of all instruction clock counts, as listed in the table below, by the processor clock period. An 8 MHz processor clock has a clock period of 125 nanoseconds and requires an 80286 system clock (CLK input) of 16 MHz.

Instruction Clock Count Assumptions

1. The instruction has been prefetched, decoded, and is ready for execution. Control transfer instruction clock counts include all time required to fetch, decode, and prepare the next instruction for execution.
2. Bus cycles do not require wait states.
3. There are no processor extension data transfer or local bus HOLD requests.
4. No exceptions occur during instruction execution.

Instruction Set Summary Notes

Addressing displacements selected by the MOD field are not shown. If necessary they appear after the instruction fields shown.

Above/below refers to unsigned value

Greater refers to positive signed value

Less refers to less positive (more negative) signed values

if d = 1 then to register; if d = 0 then from register

if w = 1 then word instruction; if w = 0 then byte instruction

if s = 0 then 16-bit immediate data form the operand

if s = 1 then an immediate data byte is sign-extended to form the 16-bit operand

x don't care

z used for string primitives for comparison with ZF FLAG

If two clock counts are given, the smaller refers to a register operand and the larger refers to a memory operand

* = add one clock if offset calculation requires summing 3 elements

n = number of times repeated

m = number of bytes of code in next instruction

Level (L)—Lexical nesting level of the procedure

The following comments describe possible exceptions, side effects, and allowed usage for instructions in both operating modes of the 80286.

REAL ADDRESS MODE ONLY

1. This is a protected mode instruction. Attempted execution in real address mode will result in an undefined opcode exception (6).
2. A segment overrun exception (13) will occur if a word operand reference at offset FFFF(H) is attempted.
3. This instruction may be executed in real address mode to initialize the CPU for protected mode.
4. The IOPL and NT fields will remain 0.
5. Processor extension segment overrun interrupt (9) will occur if the operand exceeds the segment limit.

EITHER MODE

6. An exception may occur, depending on the value of the operand.
7. $\overline{\text{LOCK}}$ is automatically asserted regardless of the presence or absence of the LOCK instruction prefix.
8. $\overline{\text{LOCK}}$ does not remain active between all operand transfers.

PROTECTED VIRTUAL ADDRESS MODE ONLY

9. A general protection exception (13) will occur if the memory operand can not be used due to either a segment limit or access rights violation. If a stack segment limit is violated, a stack segment overrun exception (12) occurs.
10. For segment load operations, the CPL, RPL, and DPL must agree with privilege rules to avoid an exception. The segment must be present to avoid a not-present exception (11). If the SS register is the destination, and a segment not-present violation occurs, a stack exception (12) occurs.
11. All segment descriptor accesses in the GDT or LDT made by this instruction will automatically assert $\overline{\text{LOCK}}$ to maintain descriptor integrity in multiprocessor systems.
12. JMP, CALL, INT, RET, IRET instructions referring to another code segment will cause a general protection exception (13) if any privilege rule is violated.
13. A general protection exception (13) occurs if CPL $\neq$ 0.
14. A general protection exception (13) occurs if CPL > IOPL.
15. The IF field of the flag word is not updated if CPL > IOPL. The IOPL field is updated only if CPL = 0.
16. Any violation of privilege rules as applied to the selector operand do not cause a protection exception; rather, the instruction does not return a result and the zero flag is cleared.
17. If the starting address of the memory operand violates a segment limit, or an invalid access is attempted, a general protection exception (13) will occur before the ESC instruction is executed. A stack segment overrun exception (12) will occur if the stack limit is violated by the operand's starting address. If a segment limit is violated during an attempted data transfer then a processor extension segment overrun exception (9) occurs.
18. The destination of an INT, JMP, CALL, RET or IRET instruction must be in the defined limit of a code segment or a general protection exception (13) will occur.

80286 INSTRUCTION SET SUMMARY

FUNCTION	FORMAT				CLOCK COUNT		COMMENTS	
					Real Address Mode	Protected Virtual Address Mode	Real Address Mode	Protected Virtual Address Mode
DATA TRANSFER								
MOV = Move:								
Register to Register/Memory	1 0 0 0 1 0 0 w	mod reg r/m			2,3*	2,3*	2	9
Register/memory to register	1 0 0 0 1 0 1 w	mod reg r/m			2,5*	2,5*	2	9
Immediate to register/memory	1 1 0 0 0 1 1 w	mod 0 0 0 r/m	data	data if w = 1	2,3*	2,3*	2	9
Immediate to register	1 0 1 1 w reg	data	data if w = 1		2	2		
Memory to accumulator	1 0 1 0 0 0 0 w	addr-low	addr-high		5	5	2	9
Accumulator to memory	1 0 1 0 0 0 1 w	addr-low	addr-high		3	3	2	9
Register/memory to segment register	1 0 0 0 1 1 1 0	mod 0 reg r/m			2,5*	17,19*	2	9,10,11
Segment register to register/memory	1 0 0 0 1 1 0 0	mod 0 reg r/m			2,3*	2,3*	2	9
PUSH = Push:								
Memory	1 1 1 1 1 1 1 1	mod 1 1 0 r/m			5*	5*	2	9
Register	0 1 0 1 0 reg				3	3	2	9
Segment register	0 0 0 reg 1 1 0				3	3	2	9
Immediate	0 1 1 0 1 0 s 0	data	data if s = 0		3	3	2	9
PUSHA = Push All	0 1 1 0 0 0 0 0				17	17	2	9
POP = Pop:								
Memory	1 0 0 0 1 1 1 1	mod 0 0 0 r/m			5*	5*	2	9
Register	0 1 0 1 1 reg				5	5	2	9
Segment register	0 0 0 reg 1 1 1	(reg ≠ 01)			5	20	2	9,10,11
POPA = Pop All	0 1 1 0 0 0 0 1				19	19	2	9
XCHG = Exchange:								
Register/memory with register	1 0 0 0 0 1 1 w	mod reg r/m			3,5*	3,5*	2,7	7,9
Register with accumulator	1 0 0 1 0 reg				3	3		
IN = Input from:								
Fixed port	1 1 1 0 0 1 0 w	port			5	5		14
Variable port	1 1 1 0 1 1 0 w				5	5		14
OUT = Output to:								
Fixed port	1 1 1 0 0 1 1 w	port			3	3		14
Variable port	1 1 1 0 1 1 1 w				3	3		14
XLAT = Translate byte to AL	1 1 0 1 0 1 1 1				5	5		9
LEA = Load EA to register	1 0 0 0 1 1 0 1	mod reg r/m			3*	3*		
LDS = Load pointer to DS	1 1 0 0 0 1 0 1	mod reg r/m	(mod ≠ 11)		7*	21*	2	9,10,11
LES = Load pointer to ES	1 1 0 0 0 1 0 0	mod reg r/m	(mod ≠ 11)		7*	21*	2	9,10,11
LAHF = Load AH with flags	1 0 0 1 1 1 1 1				2	2		
SAHF = Store AH into flags	1 0 0 1 1 1 1 0				2	2		
PUSHF = Push flags	1 0 0 1 1 1 0 0				3	3	2	9
POPF = Pop flags	1 0 0 1 1 1 0 1				5	5	2,4	9,15

Shaded areas indicate instructions not available in iAPX 86, 88 microsystems.

80286 INSTRUCTION SET SUMMARY (Continued)

FUNCTION	FORMAT	CLOCK COUNT: Real Address Mode	CLOCK COUNT: Protected Virtual Address Mode	COMMENTS: Real Address Mode	COMMENTS: Protected Virtual Address Mode
ARITHMETIC					
ADD = Add:					
Reg/memory with register to either	0 0 0 0 0 0 d w \| mod reg r/m	2,7*	2,7*	2	9
Immediate to register/memory	1 0 0 0 0 0 s w \| mod 0 0 0 r/m \| data \| data if s w = 01	3,7*	3,7*	2	9
Immediate to accumulator	0 0 0 0 0 1 0 w \| data \| data if w = 1	3	3		
ADC = Add with carry:					
Reg/memory with register to either	0 0 0 1 0 0 d w \| mod reg r/m	2,7*	2,7*	2	9
Immediate to register/memory	1 0 0 0 0 0 s w \| mod 0 1 0 r/m \| data \| data if s w = 01	3,7*	3,7*	2	9
Immediate to accumulator	0 0 0 1 0 1 0 w \| data \| data if w = 1	3	3		
INC = Increment:					
Register/memory	1 1 1 1 1 1 1 w \| mod 0 0 0 r/m	2,7*	2,7*	2	9
Register	0 1 0 0 0 reg	2	2		
SUB = Subtract:					
Reg/memory and register to either	0 0 1 0 1 0 d w \| mod reg r/m	2,7*	2,7*	2	9
Immediate from register/memory	1 0 0 0 0 0 s w \| mod 1 0 1 r/m \| data \| data if s w = 01	3,7*	3,7*	2	9
Immediate from accumulator	0 0 1 0 1 1 0 w \| data \| data if w = 1	3	3		
SBB = Subtract with borrow:					
Reg/memory and register to either	0 0 0 1 1 0 d w \| mod reg r/m	2,7*	2,7*	2	9
Immediate from register/memory	1 0 0 0 0 0 s w \| mod 0 1 1 r/m \| data \| data if s w = 01	3,7*	3,7*	2	9
Immediate from accumulator	0 0 0 1 1 1 0 w \| data \| data if w = 1	3	3		
DEC = Decrement:					
Register/memory	1 1 1 1 1 1 1 w \| mod 0 0 1 r/m	2,7*	2,7*	2	9
Register	0 1 0 0 1 reg	2	2		
CMP = Compare:					
Register/memory with register	0 0 1 1 1 0 1 w \| mod reg r/m	2,6*	2,6*	2	9
Register with register/memory	0 0 1 1 1 0 0 w \| mod reg r/m	2,7*	2,7*	2	9
Immediate with register/memory	1 0 0 0 0 0 s w \| mod 1 1 1 r/m \| data \| data if s w = 01	3,6*	3,6*	2	9
Immediate with accumulator	0 0 1 1 1 1 0 w \| data \| data if w = 1	3	3		
NEG = Change sign	1 1 1 1 0 1 1 w \| mod 0 1 1 r/m	2	7*	2	7
AAA = ASCII adjust for add	0 0 1 1 0 1 1 1	3	3		
DAA = Decimal adjust for add	0 0 1 0 0 1 1 1	3	3		
AAS = ASCII adjust for subtract	0 0 1 1 1 1 1 1	3	3		
DAS = Decimal adjust for subtract	0 0 1 0 1 1 1 1	3	3		
MUL = Multiply (unsigned):	1 1 1 1 0 1 1 w \| mod 1 0 0 r/m				
Register-Byte		13	13		
Register-Word		21	21		
Memory-Byte		16*	16*	2	9
Memory-Word		24*	24*	2	9
IMUL = Integer multiply (signed):	1 1 1 1 0 1 1 w \| mod 1 0 1 r/m				
Register-Byte		13	13		
Register-Word		21	21		
Memory-Byte		16*	16*	2	9
Memory-Word		24*	24*	2	9
IMUL = Integer immediate multiply (signed)	0 1 1 0 1 0 s 1 \| mod reg r/m \| data \| data if s = 0	21,24*	21,24*	2	9
DIV = Divide (unsigned):	1 1 1 1 0 1 1 w \| mod 1 1 0 r/m				
Register-Byte		14	14	6	6
Register-Word		22	22	6	6
Memory-Byte		17*	17*	2,6	6,9
Memory-Word		25*	25*	2,6	6,9

Shaded areas indicate instructions not available in iAPX 86, 88 microsystems.

FUNCTION	FORMAT	CLOCK COUNT		COMMENTS	
		Real Address Mode	Protected Virtual Address Mode	Real Address Mode	Protected Virtual Address Mode
ARITHMETIC (Continued):					
IDIV = Integer divide (signed):	1 1 1 1 0 1 1 w \| mod 1 1 1 r/m				
Register-Byte		17	17	6	6
Register-Word		25	25	6	6
Memory-Byte		20*	20*	2,6	6,9
Memory-Word		28*	28*	2,6	6,9
AAM = ASCII adjust for multiply	1 1 0 1 0 1 0 0 \| 0 0 0 0 1 0 1 0	16	16		
AAD = ASCII adjust for divide	1 1 0 1 0 1 0 1 \| 0 0 0 0 1 0 1 0	14	14		
CBW = Convert byte to word	1 0 0 1 1 0 0 0	2	2		
CWD = Convert word to double word	1 0 0 1 1 0 0 1	2	2		
LOGIC					
Shift/Rotate Instructions:					
Register/Memory by 1	1 1 0 1 0 0 0 w \| mod TTT r/m	2,7*	2,7*	2	9
Register/Memory by CL	1 1 0 1 0 0 1 w \| mod TTT r/m	5+n,8+n*	5+n,8+n*	2	9
Register/Memory by Count	1 1 0 0 0 0 0 w \| mod TTT r/m \| count	5+n,8+n*	5+n,8+n*	2	9
	TTT Instruction: 0 0 0 ROL; 0 0 1 ROR; 0 1 0 RCL; 0 1 1 RCR; 1 0 0 SHL/SAL; 1 0 1 SHR; 1 1 1 SAR				
AND = And:					
Reg/memory and register to either	0 0 1 0 0 0 d w \| mod reg r/m	2,7*	2,7*	2	9
Immediate to register/memory	1 0 0 0 0 0 0 w \| mod 1 0 0 r/m \| data \| data if w = 1	3,7*	3,7*	2	9
Immediate to accumulator	0 0 1 0 0 1 0 w \| data \| data if w = 1	3	3		
TEST = And function to flags, no result:					
Register/memory and register	1 0 0 0 0 1 0 w \| mod reg r/m	2,6*	2,6*	2	9
Immediate data and register/memory	1 1 1 1 0 1 1 w \| mod 0 0 0 r/m \| data \| data if w = 1	3,6*	3,6*	2	9
Immediate data and accumulator	1 0 1 0 1 0 0 w \| data \| data if w = 1	3	3		
OR = Or:					
Reg/memory and register to either	0 0 0 0 1 0 d w \| mod reg r/m	2,7*	2,7*	2	9
Immediate to register/memory	1 0 0 0 0 0 0 w \| mod 0 0 1 r/m \| data \| data if w = 1	3,7*	3,7*	2	9
Immediate to accumulator	0 0 0 0 1 1 0 w \| data \| data if w = 1	3	3		
XOR = Exclusive or:					
Reg/memory and register to either	0 0 1 1 0 0 d w \| mod reg r/m	2,7*	2,7*	2	9
Immediate to register/memory	1 0 0 0 0 0 0 w \| mod 1 1 0 r/m \| data \| data if w = 1	3,7*	3,7*	2	9
Immediate to accumulator	0 0 1 1 0 1 0 w \| data \| data if w = 1	3	3		
NOT = Invert register/memory	1 1 1 1 0 1 1 w \| mod 0 1 0 r/m	2,7*	2,7*	2	9
STRING MANIPULATION:					
MOVS = Move byte/word	1 0 1 0 0 1 0 w	5	5	2	9
CMPS = Compare byte/word	1 0 1 0 0 1 1 w	8	8	2	9
SCAS = Scan byte/word	1 0 1 0 1 1 1 w	7	7	2	9
LODS = Load byte/wd to AL/AX	1 0 1 0 1 1 0 w	5	5	2	9
STOS = Stor byte/wd from AL/A	1 0 1 0 1 0 1 w	3	3	2	9
INS = Input byte/wd from DX port	0 1 1 0 1 1 0 w	5	5	2	9,14
OUTS = Output byte/wd to DX port	0 1 1 0 1 1 1 w	5	5	2	9,14

Shaded areas indicate instructions not available in iAPX 86, 88 microsystems.

80286 INSTRUCTION SET SUMMARY (Continued)

FUNCTION	FORMAT	CLOCK COUNT: Real Address Mode	CLOCK COUNT: Protected Virtual Address Mode	COMMENTS: Real Address Mode	COMMENTS: Protected Virtual Address Mode
STRING MANIPULATION (Continued):					
Repeated by count in CX					
MOVS = Move string	1 1 1 1 0 0 1 0 \| 1 0 1 0 0 1 0 w	5+4n	5+4n	2	9
CMPS = Compare string	1 1 1 1 0 0 1 z \| 1 0 1 0 0 1 1 w	5+9n	5+9n	2,8	8,9
SCAS = Scan string	1 1 1 1 0 0 1 z \| 1 0 1 0 1 1 1 w	5+8n	5+8n	2,8	8,9
LODS = Load string	1 1 1 1 0 0 1 0 \| 1 0 1 0 1 1 0 w	5+4n	5+4n	2,8	8,9
STOS = Store string	1 1 1 1 0 0 1 0 \| 1 0 1 0 1 0 1 w	4+3n	4+3n	2,8	8,9
INS = Input string	1 1 1 1 0 0 1 0 \| 0 1 1 0 1 1 0 w	5+4n	5+4n	2	9,14
OUTS = Output string	1 1 1 1 0 0 1 0 \| 0 1 1 0 1 1 1 w	5+4n	5+4n	2	9,14
CONTROL TRANSFER					
CALL = Call:					
Direct within segment	1 1 1 0 1 0 0 0 \| disp-low \| disp-high	7+m	7+m	2	18
Register/memory indirect within segment	1 1 1 1 1 1 1 1 \| mod 0 1 0 r/m	7+m,11+m*	7+m,11+m*	2,8	8,9,18
Direct intersegment	1 0 0 1 1 0 1 0 \| segment offset \| segment selector	13+m	26+m	2	11,12,18
Protected Mode Only (Direct intersegment):					
Via call gate to same privilege level			41+m		8,11,12,18
Via call gate to different privilege level, no parameters			82+m		8,11,12,18
Via call gate to different privilege level, x parameters			86+4x+m		8,11,12,18
Via TSS			177+m		8,11,12,18
Via task gate			182+m		8,11,12,18
Indirect intersegment	1 1 1 1 1 1 1 1 \| mod 0 1 1 r/m (mod ≠ 11)	16+m	29+m*	2	8,9,11,12,18
Protected Mode Only (Indirect intersegment):					
Via call gate to same privilege level			44+m*		8,9,11,12,18
Via call gate to different privilege level, no parameters			83+m*		8,9,11,12,18
Via call gate to different privilege level, x parameters			90+4x+m*		8,9,11,12,18
Via TSS			180+m*		8,9,11,12,18
Via task gate			185+m*		8,9,11,12,18
JMP = Unconditional jump:					
Short/long	1 1 1 0 1 0 1 1 \| disp-low	7+m	7+m		18
Direct within segment	1 1 1 0 1 0 0 1 \| disp-low \| disp-high	7+m	7+m		18
Register/memory indirect within segment	1 1 1 1 1 1 1 1 \| mod 1 0 0 r/m	7+m,11+m*	7+m,11+m*	2	9,18
Direct intersegment	1 1 1 0 1 0 1 0 \| segment offset \| segment selector	11+m	23+m		11,12,18
Protected Mode Only (Direct intersegment):					
Via call gate to same privilege level			38+m		8,11,12,18
Via TSS			175+m		8,11,12,18
Via task gate			180+m		8,11,12,18
Indirect intersegment	1 1 1 1 1 1 1 1 \| mod 1 0 1 r/m (mod ≠ 11)	15+m*	26+m*	2	8,9,11,12,18
Protected Mode Only (Indirect intersegment):					
Via call gate to same privilege level			41+m*		8,9,11,12,18
Via TSS			178+m*		8,9,11,12,18
Via task gate			183+m*		8,9,11,12,18
RET = Return from CALL:					
Within segment	1 1 0 0 0 0 1 1	11+m	11+m	2	8,9,18
Within seg adding immed to SP	1 1 0 0 0 0 1 0 \| data-low \| data-high	11+m	11+m	2	8,9,18
Intersegment	1 1 0 0 1 0 1 1	15+m	25+m	2	8,9,11,12,18
Intersegment adding immediate to SP	1 1 0 0 1 0 1 0 \| data-low \| data-high	15+m		2	8,9,11,12,18
Protected Mode Only (RET):					
To different privilege level			55+m		9,11,12,18

Shaded areas indicate instructions not available in iAPX 86, 88 microsystems.

80286 INSTRUCTION SET SUMMARY (Continued)

FUNCTION	FORMAT	CLOCK COUNT: Real Address Mode	CLOCK COUNT: Protected Virtual Address Mode	COMMENTS: Real Address Mode	COMMENTS: Protected Virtual Address Mode
CONTROL TRANSFER (Continued):					
JE/JZ = Jump on equal/zero	0 1 1 1 0 1 0 0 \| disp	7+m or 3	7+m or 3		18
JL/JNGE = Jump on less/not greater or equal	0 1 1 1 1 1 0 0 \| disp	7+m or 3	7+m or 3		18
JLE/JNG = Jump on less or equal/not greater	0 1 1 1 1 1 1 0 \| disp	7+m or 3	7+m or 3		18
JB/JNAE = Jump on below/not above or equal	0 1 1 1 0 0 1 0 \| disp	7+m or 3	7+m or 3		18
JBE/JNA = Jump on below or equal/not above	0 1 1 1 0 1 1 0 \| disp	7+m or 3	7+m or 3		18
JP/JPE = Jump on parity/parity even	0 1 1 1 1 0 1 0 \| disp	7+m or 3	7+m or 3		18
JO = Jump on overflow	0 1 1 1 0 0 0 0 \| disp	7+m or 3	7+m or 3		18
JS = Jump on sign	0 1 1 1 1 0 0 0 \| disp	7+m or 3	7+m or 3		18
JNE/JNZ = Jump on not equal/not zero	0 1 1 1 0 1 0 1 \| disp	7+m or 3	7+m or 3		18
JNL/JGE = Jump on not less/greater or equal	0 1 1 1 1 1 0 1 \| disp	7+m or 3	7+m or 3		18
JNLE/JG = Jump on not less or equal/greater	0 1 1 1 1 1 1 1 \| disp	7+m or 3	7+m or 3		18
JNB/JAE = Jump on not below/above or equal	0 1 1 1 0 0 1 1 \| disp	7+m or 3	7+m or 3		18
JNBE/JA = Jump on not below or equal/above	0 1 1 1 0 1 1 1 \| disp	7+m or 3	7+m or 3		18
JNP/JPO = Jump on not par/par odd	0 1 1 1 1 0 1 1 \| disp	7+m or 3	7+m or 3		18
JNO = Jump on not overflow	0 1 1 1 0 0 0 1 \| disp	7+m or 3	7+m or 3		18
JNS = Jump on not sign	0 1 1 1 1 0 0 1 \| disp	7+m or 3	7+m or 3		18
LOOP = Loop CX times	1 1 1 0 0 0 1 0 \| disp	8+m or 4	8+m or 4		18
LOOPZ/LOOPE = Loop while zero/equal	1 1 1 0 0 0 0 1 \| disp	8+m or 4	8+m or 4		18
LOOPNZ/LOOPNE = Loop while not zero/equal	1 1 1 0 0 0 0 0 \| disp	8+m or 4	8+m or 4		18
JCXZ = Jump on CX zero	1 1 1 0 0 0 1 1 \| disp	8+m or 4	8+m or 4		18
ENTER = Enter Procedure	1 1 0 0 1 0 0 0 \| data-low \| data-high \| L				
L=0		11	11	2,8	8,9
L=1		15	15	2,8	8,9
L>1		16+4(L−1)	16+4(L−1)	2,8	8,9
LEAVE = Leave Procedure	1 1 0 0 1 0 0 1	5	5	2,8	8,9
INT = Interrupt:					
Type specified	1 1 0 0 1 1 0 1 \| type	23+m		2,7,8	
Type 3	1 1 0 0 1 1 0 0	23+m		2,7,8	
INTO = Interrupt on overflow	1 1 0 0 1 1 1 0	24+m or 3 (3 if no interrupt)	(3 if no interrupt)	2,6,8	
Protected Mode Only:					
Via interrupt or trap gate to same privilege level			40+m		7,8,11,12,18
Via interrupt or trap gate to fit different privilege level			78+m		7,8,11,12,18
Via Task Gate			167+m		7,8,11,12,18
IRET = Interrupt return	1 1 0 0 1 1 1 1	17+m	31+m	2,4	8,9,11,12,15,18
Protected Mode Only:					
To different privilege level			55+m		8,9,11,12,15,18
To different task (NT = 1)			169+m		8,9,11,12,18
BOUND = Detect value out of range	0 1 1 0 0 0 1 0 \| mod reg r/m	13*	13* (Use INT clock count if exception 5)	2,6	8,8,9,11,12,18

Shaded areas indicate instructions not available in iAPX 86, 88 microsystems.

80286 INSTRUCTION SET SUMMARY (Continued)

FUNCTION	FORMAT	CLOCK COUNT: Real Address Mode	CLOCK COUNT: Protected Virtual Address Mode	COMMENTS: Real Address Mode	COMMENTS: Protected Virtual Address Mode
PROCESSOR CONTROL					
CLC = Clear carry	1 1 1 1 1 0 0 0	2	2		
CMC = Complement carry	1 1 1 1 0 1 0 1	2	2		
STC = Set carry	1 1 1 1 1 0 0 1	2	2		
CLD = Clear direction	1 1 1 1 1 1 0 0	2	2		
STD = Set direction	1 1 1 1 1 1 0 1	2	2		
CLI = Clear interrupt	1 1 1 1 1 0 1 0	3	3		14
STI = Set interrupt	1 1 1 1 1 0 1 1	2	2		14
HLT = Halt	1 1 1 1 0 1 0 0	2	2		13
WAIT = Wait	1 0 0 1 1 0 1 1	3	3		
LOCK = Bus lock prefix	1 1 1 1 0 0 0 0	0	0		14
CTS = Clear task switched flag	0 0 0 0 1 1 1 1 \| 0 0 0 0 0 1 1 0	2	2	3	13
ESC = Processor Extension Escape	1 1 0 1 1 T T T \| mod LLL r/m (TTT LLL are opcode to processor extension)	9-20*	9-20*	5,8	8,17
SEG = Segment Override Prefix	001 reg 110	0	0		
PROTECTION CONTROL					
LGDT = Load global descriptor table register	0 0 0 0 1 1 1 1 \| 0 0 0 0 0 0 0 1 \| mod 0 1 0 r/m	11*	11*	2,3	9,13
SGDT = Store global descriptor table register	0 0 0 0 1 1 1 1 \| 0 0 0 0 0 0 0 1 \| mod 0 0 0 r/m	11*	11*	2,3	9
LIDT = Load interrupt descriptor table register	0 0 0 0 1 1 1 1 \| 0 0 0 0 0 0 0 1 \| mod 0 1 1 r/m	12*	12*	2,3	9,13
SIDT = Store interrupt descriptor table register	0 0 0 0 1 1 1 1 \| 0 0 0 0 0 0 0 1 \| mod 0 0 1 r/m	12*	12*	2,3	9
LLDT = Load local descriptor table register from register memory	0 0 0 0 1 1 1 1 \| 0 0 0 0 0 0 0 0 \| mod 0 1 0 r/m		17,19*	1	9,11,13
SLDT = Store local descriptor table register to register/memory	0 0 0 0 1 1 1 1 \| 0 0 0 0 0 0 0 0 \| mod 0 0 0 r/m		2,3*	1	9
LTR = Load task register from register/memory	0 0 0 0 1 1 1 1 \| 0 0 0 0 0 0 0 0 \| mod 0 1 1 r/m		17,19*	1	9,11,13
STR = Store task register to register memory	0 0 0 0 1 1 1 1 \| 0 0 0 0 0 0 0 0 \| mod 0 0 1 r/m		2,3*	1	9
LMSW = Load machine status word from register/memory	0 0 0 0 1 1 1 1 \| 0 0 0 0 0 0 0 1 \| mod 1 1 0 r/m	3,6*	3,6*	2,3	9,13
SMSW = Store machine status word	0 0 0 0 1 1 1 1 \| 0 0 0 0 0 0 0 1 \| mod 1 0 0 r/m	2,3*	2,3*	2,3	9
LAR = Load access rights from register/memory	0 0 0 0 1 1 1 1 \| 0 0 0 0 0 0 1 0 \| mod reg r/m		14,16*	1	9,11,16
LSL = Load segment limit from register/memory	0 0 0 0 1 1 1 1 \| 0 0 0 0 0 0 1 1 \| mod reg r/m		14,16*	1	9,11,16
ARPL = Adjust requested privilege level: from register/memory	0 1 1 0 0 0 1 1 \| mod reg r/m		10*,11*	2	8,9
VERR = Verify read access: register/memory	0 0 0 0 1 1 1 1 \| 0 0 0 0 0 0 0 0 \| mod 1 0 0 r/m		14,16*	1	9,11,16
VERR = Verify write access:	0 0 0 0 1 1 1 1 \| 0 0 0 0 0 0 0 0 \| mod 1 0 1 r/m		14,16*	1	9,11,16

Shaded areas indicate instructions not available in iAPX 86, 88 microsystems.

The effective Address (EA) of the memory operand is computed according to the mod and r/m fields:

if mod = 11 then r/m is treated as a REG field
if mod = 00 then DISP = 0*, disp-low and disp-high are absent
if mod = 01 then DISP = disp-low sign-extended to 16-bits, disp-high is absent
if mod = 10 then DISP = disp-high: disp-low

if r/m = 000 then EA = (BX) + (SI) + DISP
if r/m = 001 then EA = (BX) + (DI) + DISP
if r/m = 010 then EA = (BP) + (SI) + DISP
if r/m = 011 then EA = (BP) + (DI) + DISP
if r/m = 100 then EA = (SI) + DISP
if r/m = 101 then EA = (DI) + DISP
if r/m = 110 then EA = (BP) + DISP*
if r/m = 111 then EA = (BX) + DISP

DISP follows 2nd byte of instruction (before data if required)

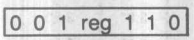
*except if mod = 00 and r/m = 110 then EA = disp-high: disp-low.

REG is assigned according to the following table:

16-Bit (w = 1)	8-Bit (w = 0)
000 AX	000 AL
001 CX	001 CL
010 DX	010 DL
011 BX	011 BL
100 SP	100 AH
101 BP	101 CH
110 SI	110 DH
111 DI	111 BH

The physical addresses of all operands addressed by the BP register are computed using the SS segment register. The physical addresses of the destination operands of the string primitive operations (those addressed by the DI register) are computed using the ES segment, which may not be overridden.

SEGMENT OVERRIDE PREFIX



reg is assigned according to the following:

reg	Segment Register
00	ES
01	CS
10	SS
11	DS

80287
80-Bit HMOS
NUMERIC PROCESSOR EXTENSION

- **High Performance 80-Bit Internal Architecture**
- **Implements Proposed IEEE Floating Point Standard 754**
- **Expands iAPX 286/10 Datatypes to Include 32-, 64-, 80-Bit Floating Point, 32-, 64-Bit Integers and 18-Digit BCD Operands**
- **Object Code Compatible with 8087**
- **Built-in Exception Handling**
- **Operates in Both Real and Protected Mode iAPX 286 Systems**
- **Protected Mode Operation Completely Conforms to the iAPX 286 Memory Management and Protection Mechanisms**
- **Directly Extends iAPX 286/10 Instruction Set to Trigonometric, Logarithmic, Exponential and Arithmetic Instructions for All Datatypes**
- **8x80-Bit, Individually Addressable, Numeric Register Stack**
- **Available in EXPRESS—Standard Temperature Range**

The Intel® 80287 is a high performance numerics processor extension that extends the iAPX 286/10 architecture with floating point, extended integer and BCD data types. The iAPX 286/20 computing system (80286 with 80287) fully conforms to the proposed IEEE Floating Point Standard. Using a numerics oriented architecture, the 80287 adds over fifty mnemonics to the iAPX 286/20 instruction set, making the iAPX 286/20 a complete solution for high performance numeric processing. The 80287 is impiemented in N-channel, depletion load, silicon gate technology (HMOS) and packaged in a 40-pin ceramic package. The iAPX 286/20 is object code compatible with the iAPX 86/20 and iAPX 88/20.

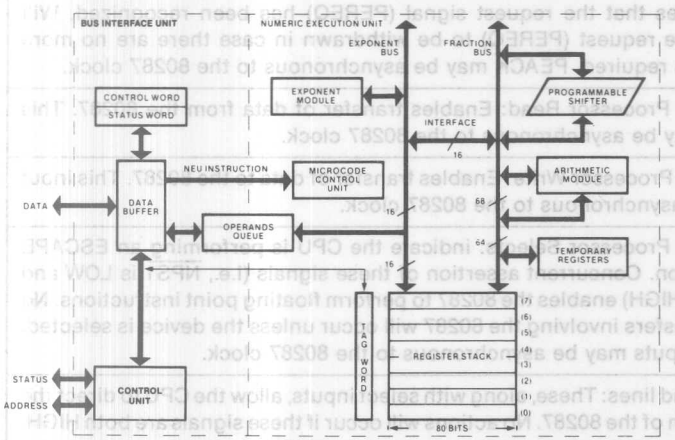
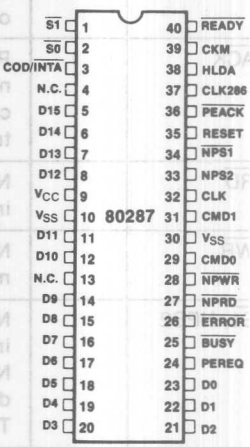


Figure 1. 80287 Block Diagram



NOTE:
N.C. PINS MUST NOT BE CONNECTED.

Figure 2. 80287 Pin Configuration

Intel Corporation Assumes No Responsibility for the Use of Any Circuitry Other Than Circuitry Embodied in an Intel Product. No Other Circuit Patent Licenses are Implied.
© INTEL CORPORATION, 1983.

FEBRUARY 1983
ORDER NUMBER: 210920-001

Table 1. 80287 Pin Description

Symbols	Type	Name and Function
CLK	I	Clock input: this clock provides the basic timing for internal 80287 operations. Special MOS level inputs are required. The 82284 or 8284A CLK outputs are compatible to this input.
CKM	I	Clock Mode signal: indicates whether CLK input is to be divided by 3 or used directly. A HIGH input will select the latter option. This input may be connected to V_{CC} or V_{SS} as appropriate. This input must be either HIGH or LOW 20 CLK cycles before RESET goes LOW.
RESET	I	System Reset: causes the 80287 to immediately terminate its present activity and enter a dormant state. RESET is required to be HIGH for more than 4 80287 CLK cycles. For proper initialization the HIGH-LOW transition must occur no sooner than 50 μs after V_{CC} and CLK meet their D.C. and A.C. specifications.
D15-D0	I/O	Data: 16-bit bidirectional data bus. Inputs to these pins may be applied asynchronous to the 80287 clock.
$\overline{\text{BUSY}}$	O	Busy status: asserted by the 80287 to indicate that it is currently executing a command.
$\overline{\text{ERROR}}$	O	Error status: reflects the ES bit of the status word. This signal indicates that an unmasked error condition exists.
PEREQ	O	Processor Extension Data Channel operand transfer request: a HIGH on this output indicates that the 80287 is ready to transfer data. PEREQ will be disabled upon assertion of $\overline{\text{PEACK}}$ or upon actual data transfer, whichever occurs first, if no more transfers are required.
$\overline{\text{PEACK}}$	I	Processor Extension Data Channel operand transfer ACKnowledge: acknowledges that the request signal (PEREQ) has been recognized. Will cause the request (PEREQ) to be withdrawn in case there are no more transfers required. $\overline{\text{PEACK}}$ may be asynchronous to the 80287 clock.
$\overline{\text{NPRD}}$	I	Numeric Processor Read: Enables transfer of data from the 80287. This input may be asynchronous to the 80287 clock.
$\overline{\text{NPWR}}$	I	Numeric Processor Write: Enables transfer of data to the 80287. This input may be asynchronous to the 80287 clock.
$\overline{\text{NPS1}}$, NPS2	I	Numeric Processor Selects: indicate the CPU is performing an ESCAPE instruction. Concurrent assertion of these signals (i.e., $\overline{\text{NPS1}}$ is LOW and NPS2 is HIGH) enables the 80287 to perform floating point instructions. No data transfers involving the 80287 will occur unless the device is selected. These inputs may be asynchronous to the 80287 clock.
CMD1, CMD0	I	Command lines: These, along with select inputs, allow the CPU to direct the operation of the 80287. No actions will occur if these signals are both HIGH. These inputs may be asynchronous to the 80287 clock.

210920-001

Table 1. 80287 Pin Description (cont.)

Symbols	Type	Name and Function
CLK286	I	CPU Clock: This input provides a sampling edge for the 80287 inputs $\overline{S1}$, $\overline{S0}$, COD/$\overline{INTA}$, $\overline{READY}$, and HLDA. It must be connected to the 80286 CLK input.
$\overline{S1}$, $\overline{S0}$ COD/$\overline{INTA}$	I	Status: These inputs allow the 80287 to monitor the execution of ESCAPE instructions by the 80286. They must be connected to the corresponding 80286 pins.
HLDA	I	Hold Acknowledge: This input informs the 80287 when the 80286 controls the local bus. It must be connected to the 80286 HLDA output.
$\overline{READY}$	I	Ready: The end of a bus cycle is signaled by this input. It must be connected to the 80286 $\overline{READY}$ input.
V_{SS}	I	System ground, both pins must be connected to ground.
V_{CC}	I	+5V supply

FUNCTIONAL DESCRIPTION

The 80287 Numeric Processor Extension (NPX) provides arithmetic instructions for a variety of numeric data types in iAPX 286/20 systems. It also executes numerous built-in transcendental functions (e.g., tangent and log functions). The 80287 executes instructions in parallel with a 80286. It effectively extends the register and instruction set of an iAPX 286/10 system for existing iAPX 286 data types and adds several new data types as well. Figure 3 presents the program visible register model of the iAPX 286/20. Essentially, the 80287 can be treated as an additional resource or an extension to the iAPX 286/10 that can be used as a single unified system, the iAPX 286/20.

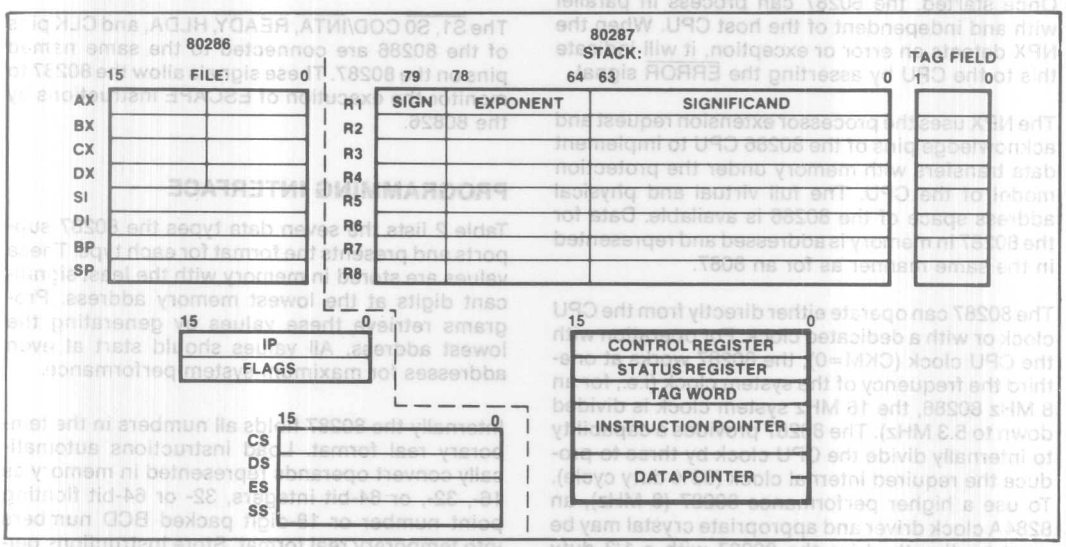


Figure 3. iAPX 286/20 Architecture

210920-001

The 80287 has two operating modes similar to the two modes of the 80286. When reset, 80287 is in the real address mode. It can be placed in the protected virtual address mode by executing the SETPM ESC instruction. The 80287 cannot be switched back to the real address mode except by reset. In the real address mode, the iAPX 286/20 is completely software compatible with iAPX 86/20, 88/20.

Once in protected mode, all references to memory for numerics data or status information, obey the iAPX 286 memory management and protection rules giving a fully protected extension of the 80286 CPU. In the protected mode, iAPX 286/20 numerics software is also completely compatible with iAPX 86/20 and iAPX 88/20.

SYSTEM CONFIGURATION

As a processor extension to an 80286, the 80287 can be connected to the CPU as shown in Figure 4. The data channel control signals (PEREQ, PEACK), the BUSY signal and the NPRD, NPWR signals, allow the NPX to receive instructions and data from the CPU. When in the protected mode, all information received by the NPX is validated by the 80286 memory management and protection unit. Once started, the 80287 can process in parallel with and independent of the host CPU. When the NPX detects an error or exception, it will indicate this to the CPU by asserting the ERROR signal.

The NPX uses the processor extension request and acknowledge pins of the 80286 CPU to implement data transfers with memory under the protection model of the CPU. The full virtual and physical address space of the 80286 is available. Data for the 80287 in memory is addressed and represented in the same manner as for an 8087.

The 80287 can operate either directly from the CPU clock or with a dedicated clock. For operation with the CPU clock (CKM=0), the 80287 works at one-third the frequency of the system clock (i.e., for an 8 MHz 80286, the 16 MHz system clock is divided down to 5.3 MHz). The 80287 provides a capability to internally divide the CPU clock by three to produce the required internal clock (33% duty cycle). To use a higher performance 80287 (8 MHz), an 8284A clock driver and appropriate crystal may be used to directly drive the 80287 with a 1/3 duty cycle clock on the CLK input (CKM=1).

HARDWARE INTERFACE

Communication of instructions and data operands between the 80286 and 80287 is handled by the CMD0, CMD1, NPS1, NPS2, NPRD, and NPWR signals. I/O port addresses 00F8H, 00FAH, and 00FCH are used by the 80286 for this communication. When any of these addresses are used, the NPS1 input must be LOW and NPS2 input HIGH. The IORC and IOWC outputs of the 82288 identify I/O space transfers (see Figure 4). CMD0 should be connected to latched 80286 A1 and CMD1 should be connected to latched 80286 A2.

I/O ports 00F8H to 00FFH are reserved for the 80286/80287 interface. To guarantee correct operation of the 80287, programs must not perform any I/O operations to these ports.

The PEREQ, PEACK, BUSY, and ERROR signals of the 80287 are connected to the same-named 80286 input. The data pins of the 80287 should be directly connected to the 80286 data bus. Note that all bus drivers connected to the 80286 local bus must be inhibited when the 80286 reads from the 80287. The use of COD/INTA and M/IO in the decoder prevents INTA bus cycles from disabling the data transceivers.

The S1, S0 COD/INTA, READY, HLDA, and CLK pins of the 80286 are connected to the same named pins on the 80287. These signals allow the 80287 to monitor the execution of ESCAPE instructions by the 80826.

PROGRAMMING INTERFACE

Table 2 lists the seven data types the 80287 supports and presents the format for each type. These values are stored in memory with the least significant digits at the lowest memory address. Programs retrieve these values by generating the lowest address. All values should start at even addresses for maximum system performance.

Internally the 80287 holds all numbers in the temporary real format. Load instructions automatically convert operands represented in memory as 16-, 32-, or 64-bit integers, 32- or 64-bit floating point number or 18-digit packed BCD numbers into temporary real format. Store instructions perform the reverse type conversion.

210920-001

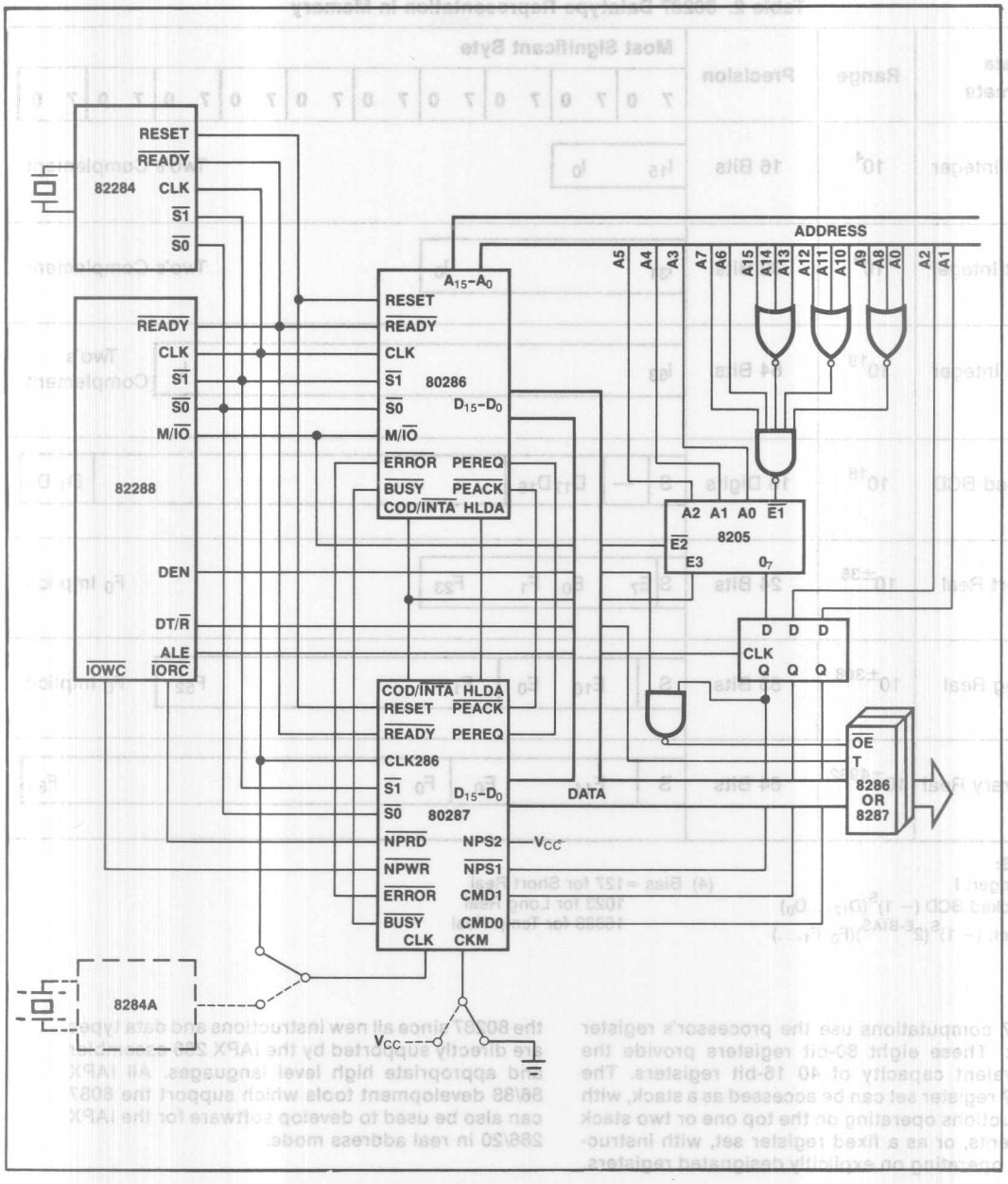


Figure 4. iAPX 286/20 System Configuration

210920-001

Table 2. 80287 Datatype Representation in Memory

Data Formats	Range	Precision	Most Significant Byte 7 0 7 0 7 0 7 0 7 0 7 0 7 0 7 0 7 0 7 0
Word Integer	10^4	16 Bits	I_{15} I_0 Two's Complement
Short Integer	10^9	32 Bits	I_{31} I_0 Two's Complement
Long Integer	10^{19}	64 Bits	I_{63} I_0 Two's Complement
Packed BCD	10^{18}	18 Digits	S — $D_{17}D_{16}$ D_1 D_0
Short Real	$10^{\pm 38}$	24 Bits	S E_7 E_0 F_1 F_{23} F_0 Implicit
Long Real	$10^{\pm 308}$	53 Bits	S E_{10} E_0 F_1 F_{52} F_0 Implicit
Temporary Real	$10^{\pm 4932}$	64 Bits	S E_{14} E_0 F_0 F_{63}

NOTES:
(1) Integer: I
(2) Packed BCD $(-1)^S(D_{17}...D_0)$
(3) Real: $(-1)^S(2^{E-BIAS})(F_0\ F_1...)$
(4) Bias = 127 for Short Real
1023 for Long Real
16383 for Temp Real

80287 computations use the processor's register stack. These eight 80-bit registers provide the equivalent capacity of 40 16-bit registers. The 80287 register set can be accessed as a stack, with instructions operating on the top one or two stack elements, or as a fixed register set, with instructions operating on explicitly designated registers.

Table 6 lists the 80287's instructions by class. No special programming tools are necessary to use the 80287 since all new instructions and data types are directly supported by the iAPX 286 assembler and appropriate high level languages. All iAPX 86/88 development tools which support the 8087 can also be used to develop software for the iAPX 286/20 in real address mode.

Table 3 gives the execution times of some typical numeric instructions.

210920-001

Table 3. Execution Time for Selected 80287 Instructions

Floating Point Instruction	Approximate Execution Time (μs)
	80287 (5 MHz Operation)
Add/Subtract	14/18
Multiply (single precision)	19
Multiply (extended precision)	27
Divide	39
Compare	9
Load (double precision)	10
Store (double precision)	21
Square Root	36
Tangent	90
Exponentiation	100

SOFTWARE INTERFACE

The iAPX 286/20 is programmed as a single processor. All communication between the 80286 and the 80287 is transparent to software. The CPU automatically controls the 80287 whenever a numeric instruction is executed. All memory addressing modes, physical memory, and virtual memory of the CPU are available for use by the NPX.

Since the NPX operates in parallel with the CPU, any errors detected by the NPX may be reported after the CPU has executed the ESCAPE instruction which caused it. To allow identification of the failing numeric instruction, the NPX contains two pointer registers which identify the address of the failing numeric instruction and the numeric memory operand if appropriate for the instruction encountering this error.

INTERRUPT DESCRIPTION

Several interrupts of the iAPX 286 are used to report exceptional conditions while executing numeric programs in either real or protected mode. The interrupts and their functions are shown in Table 4.

PROCESSOR ARCHITECTURE

As shown in Figure 1, the NPX is internally divided into two processing elements, the bus interface unit (BIU) and the numeric execution unit (NEU). The NEU executes all numeric instructions, while the BIU receives and decodes instructions, requests operand transfers to and from memory and executes processor control instructions. The two units are able to operate independently of one another allowing the BIU to maintain asynchronous communication with the CPU while the NEU is busy processing a numeric instruction.

BUS INTERFACE UNIT

The BIU decodes the ESC instruction executed by the CPU. If the ESC code defines a math instruction, the BIU transmits the formatted instruction to the NEU. If the ESC code defines an administrative instruction, the BIU executes it independently of the NEU. The parallel operation of the NPX with the CPU is normally transparant to the user. The BIU generates the $\overline{\text{BUSY}}$ and $\overline{\text{ERROR}}$ signals for 80826/80287 processor synchronization.

The 80287 executes a single numeric instruction at a time. When executing most ESC instructions, the

210920-001

Table 4. Interrupt Vectors

Interrupt Number	Interrupt Function
7	An ESC instruction was encountered when EM or TS of the 80286 MSW was set. EM=1 indicates that software emulation of the instruction is required. When TS is set, either an ESC or WAIT instruction will cause interrupt 7. This indicates that the current NPX context may not belong to the current task.
9	The second or subsequent words of a numeric operand in memory exceeded a segment's limit. This interrupt occurs after executing an ESC instruction. The saved return address will not point at the numeric instruction causing this interrupt. After processing the addressing error, the iAPX 286 program can be restarted at the return address with IRET. The address of the failing numeric instruction and numeric operand are saved in the 80287. An interrupt handler for this interrupt *must* execute FNINIT before *any* other ESC or WAIT instruction.
13	The starting address of a numeric operand is not in the segment's limit. The return address will point at the ESC instruction (including prefixes) causing this error. The 80287 has not executed this instruction. The instruction and data address in 80287 refer to a previous, correctly executed, instruction.
16	The previous numeric instruction caused an unmasked numeric error. The address of the faulty numeric instruction or numeric data operand is stored in the 80287. Only ESC or WAIT instructions can cause this interrupt. The 80286 return address will point at a WAIT or ESC instruction, including prefixes, which may be restarted after clearing the error condition in the NPX.

80286 tests the $\overline{\text{BUSY}}$ pin and waits until the 80287 indicates that it is not busy before initiating the command. Once initiated, the 80286 continues program execution while the 80287 executes the ESC instruction. In iAPX 86/20 systems, this synchronization is achieved by placing a WAIT instruction before an ESC instruction. For most ESC instructions, the iAPX 286/20 does not require a WAIT instruction before the ESC opcode. However, the iAPX 286/20 will operate correctly with these WAIT instructions. In all cases, a WAIT or ESC instruction should be inserted after any 80287 store to memory (except FSTSW and FSTCW) or load from memory (except FLDENV or FRSTOR) before the 80286 reads or changes the value.

Data transfers between memory and the 80287, when needed, are controlled by the PEREQ $\overline{\text{PEACK}}$, $\overline{\text{NPRD}}$, $\overline{\text{NPWR}}$, $\overline{\text{NPS1}}$, $\overline{\text{NPS2}}$ signals. The 80286 does the actual data transfer with memory through its processor extension data channel. Numeric data transfers with memory performed by the 80286 use the same timing as any other bus cycle. Control signals for the 80287 are generated by the 80826 as shown in Figure 4, and meet the timing requirements shown in the AC requirements section.

NUMERIC EXECUTION UNIT

The NEU executes all instructions that involve the register stack; these include arithmetic, logical, transcendental, constant and data transfer instructions. The data path in the NEU is 84 bits wide (68 fraction bits, 15 exponent bits and a sign bit) which allows internal operand transfers to be performed at very high speeds.

When the NEU begins executing an instruction, it activates the BIU $\overline{\text{BUSY}}$ signal. This signal is used in conjunction with the CPU WAIT instruction or automatically with most of the ESC instructions to synchronize both processors.

REGISTER SET

The 80287 register set is shown in Figure 5. Each of the eight data registers in the 8087's register stack

210920-001

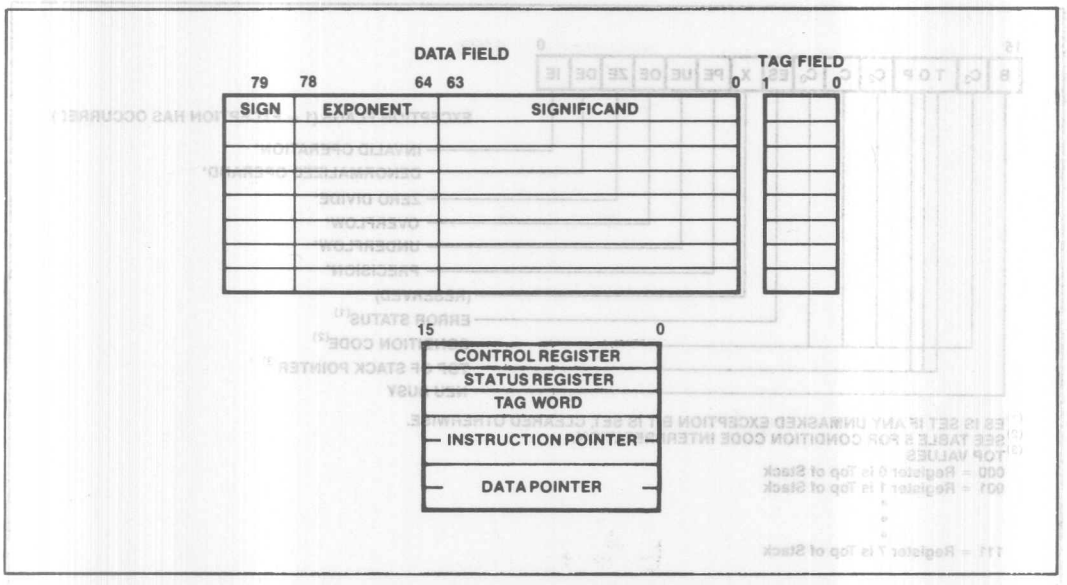


Figure 5. 80287 Register Set

is 80 bits wide and is divided into "fields" corresponding to the NPX's temporary real data type.

At a given point in time the TOP field in the status word identifies the current top-of-stack register. A "push" operation decrements TOP by 1 and loads a value into the new top register. A "pop" operation stores the value from the current top register and then increments TOP by 1. Like 80286 stacks in memory, the 80287 register stack grows "down" toward lower-addressed registers.

Instructions may address the data registers either implicitly or explicitly. Many instructions operate on the register at the top of the stack. These instructions implicitly address the register pointed by the TOP. Other instructions allow the programmer to explicitly specify the register which is to be used. Explicit register addressing is "top-relative."

STATUS WORD

The 16-bit status word (in the status register) shown in Figure 6 reflects the overall state of the 80287. It may be read and inspected by CPU code. The busy bit (bit 15) indicates whether the NEU is executing an instruction (B = 1) or is idle (B = 0).

The instructions FSTSW, FSTENV, and FSAVE which store the status word are executed exclusively by the BIU and do not set the busy bit themselves or require the Busy bit be cleared in order to be executed.

The four numeric condition code bits (C_0–C_3) are similar to the flags in a CPU: instructions that perform arithmetic operations update these bits to reflect the outcome of NDP operations. The effect of these instructions on the condition code bits is summarized in Tables 5a and 5b.

Bits 14–12 of the status word point to the 80287 register that is the current top-of-stack (TOP) as described above. Figure 6 shows the six error flags in bits 7–0 of the status word. The section on exception handling explains how they are set and used.

Bit 7 is the error status bit. This bit is set if any unmasked exception bit is set and cleared otherwise. If this bit is set, the ERROR signal is asserted.

Bits 5–0 are set to indicate that the NEU has detected an exception while executing an instruction.

210920-001

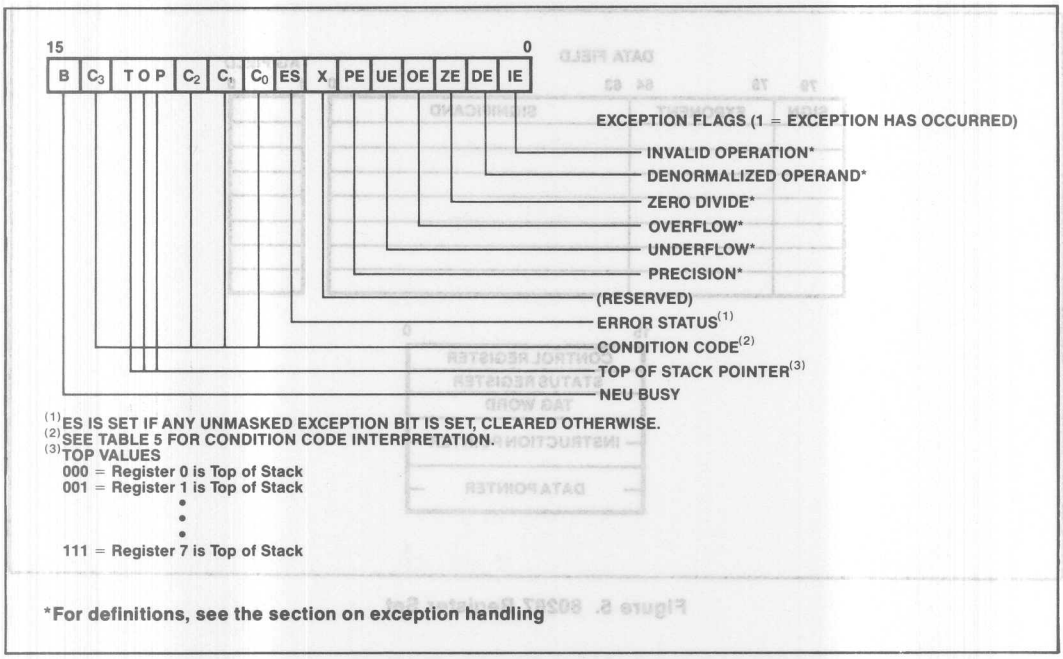


Figure 6. 80287 Status Word

TAG WORD

The tag word marks the content of each register as shown in Figure 7. The principal function of the tag word is to optimize the NPX's performance. The tag word can be used, however, to interpret the contents of 80287 registers.

INSTRUCTION AND DATA POINTERS

The instruction and data pointers (See Figures 8a and 8b) are provided for user-written error handlers. Whenever the 80287 executes a new instruction, the BIU saves the instruction address, the operand address (if present) and the instruction opcode. 80287 instructions can store this data into memory.

The instruction and data pointers appear in one of two formats depending on the operating mode of the 80287. In real mode, these values are the 20-bit physical address and 11-bit opcode formatted like the 8087. In protected mode, these values are the 32-bit virtual addresses used by the program which executed an ESC instruction. The same FLDENV/FSTENV/FSAVE/FRSTOR instructions as those of the 8087 are used to transfer these values between the 80287 registers and memory.

The saved instruction address in the 80287 will point at any prefixes which preceded the instruction. This is different than in the 8087 which only pointed at the ESCAPE instruction opcode.

CONTROL WORD

The NPX provides several processing options which are selected by loading a word from memory into the control word. Figure 9 shows the format and encoding of fields in the control word.

The low order byte of this control word configures the 80287 error and exception masking. Bits 5–0 of the control word contain individual masks for each of the six exceptions that the 80287 recognizes. The high order byte of the control word configures the 80287 operating mode including precision,

210920-001

Table 5a. Condition Code Interpretation

Instruction Type	C_3	C_2	C_1	C_0	Interpretation
Compare, Test	0	0	X	0	ST > Source or 0 (FTST)
	0	0	X	1	ST < Source or 0 (FTST)
	1	0	X	0	ST = Source or 0 (FTST)
	1	1	X	1	ST is not comparable
Remainder	Q_1	0	Q_0	Q_2	Complete reduction with three low bits of quotient (See Table 5b)
	U	1	U	U	Incomplete Reduction
Examine	0	0	0	0	Valid, positive unnormalized
	0	0	0	1	Invalid, positive, exponent =0
	0	0	1	0	Valid, negative, unnormalized
	0	0	1	1	Invalid, negative, exponent =0
	0	1	0	0	Valid, positive, normalized
	0	1	0	1	Infinity, positive
	0	1	1	0	Valid, negative, normalized
	0	1	1	1	Infinity, negative
	1	0	0	0	Zero, positive
	1	0	0	1	Empty
	1	0	1	0	Zero, negative
	1	0	1	1	Empty
	1	1	0	0	Invalid, positive, exponent = 0
	1	1	0	1	Empty
	1	1	1	0	Invalid, negative, exponent = 0
	1	1	1	1	Empty

NOTES:
1. ST = Top of stack
2. X = value is not affected by instruction
3. U = value is undefined following instruction
4. Q_n = Quotient bit n

Table 5b. Condition Code Interpretation after FPREM Instruction As a Function of Dividend Value

Dividend Range	Q_2	Q_1	Q_0
Dividend < 2 * Modulus	C_3	C_1	Q_0
Dividend < 4 * Modulus	C_3	Q_1	Q_0
Dividend ≥ 4 * Modulus	Q_2	Q_1	Q_0

NOTE:
1. Previous value of indicated bit, not affected by FPREM instruction execution.

rounding, and infinity control. The precision control bits (bits 9–8) can be used to set the 80287 internal operating precision at less than the default of temporary real (80-bit) precision. This can be useful in providing compatibility with the early generation arithmetic processors of smaller precision than the 80287. The rounding control bits (bits 11–10) provide for directed rounding and true chop as well as the unbiased round to nearest even mode specified in the IEEE standard. Control over closure of the number space at infinity is also provided (either affine closure: ± ∞, or projective closure: ∞, is treated as unsigned, may be specified).

210920-001

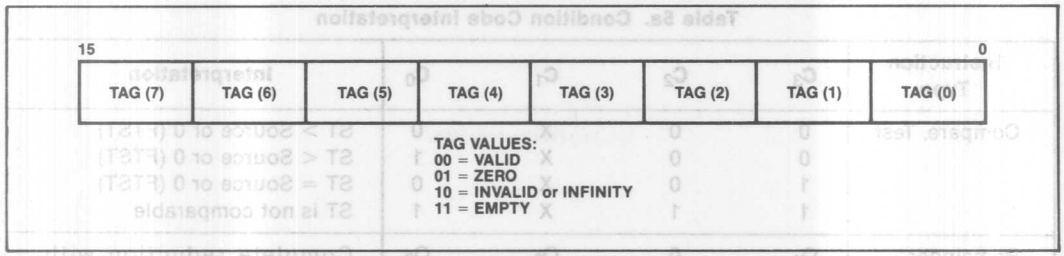


Figure 7. 80287 Tag Word

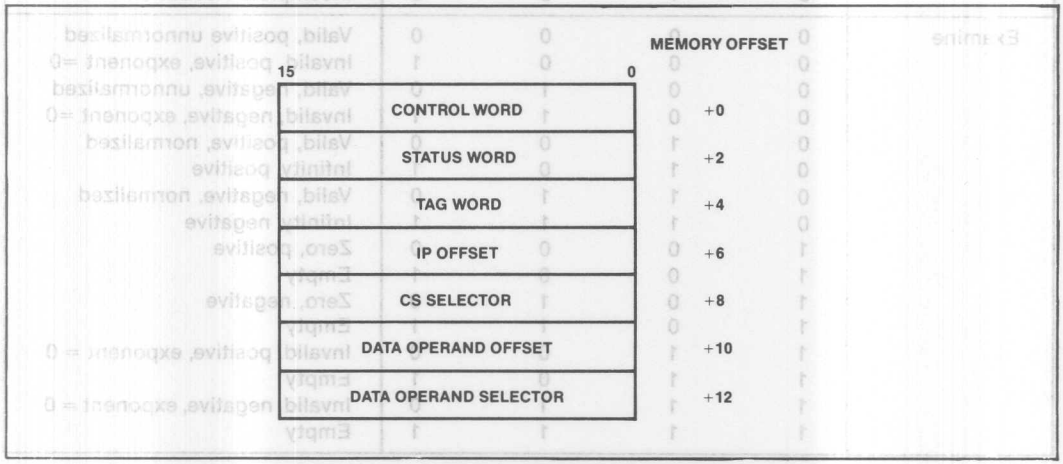


Figure 8a. Protected Mode Instruction and Data Pointer Image in Memory

EXCEPTION HANDLING

The 80287 detects six different exception conditions that can occur during instruction execution. Any or all exceptions will cause the assertion of ERROR signal if the appropriate exception masks are not set.

The exceptions that the 80287 detects and the 'default' procedures that will be carried out if the exception is masked, are as follows:

Invalid Operation: Stack overflow, stack underflow, indeterminate form (0/0, ∞-∞, etc.) or the use of a Non-Number (NAN) as an operand. An exponent value of all ones and non-zero significand is reserved to identify NANs. If this exception is masked, the 80287 default response is to generate a specific NAN called INDEFINITE, or to propogate already existing NANs as the calculation result.

Overflow: The result is too large in magnitude to fit the specified format. The 80287 will generate an encoding for infinity if this exception is masked.

Zero Divisor: The divisor is zero while the dividend is a non-infinite, non-zero number. Again, the 80287 will generate an encoding for infinity if this exception is masked.

Underflow: The result is non-zero but too small in magnitude to fit in the specified format. If this exception is masked the 82087 will denormalize (shift right) the fraction until the exponent is in range. The process is called gradual underflow.

210920-001

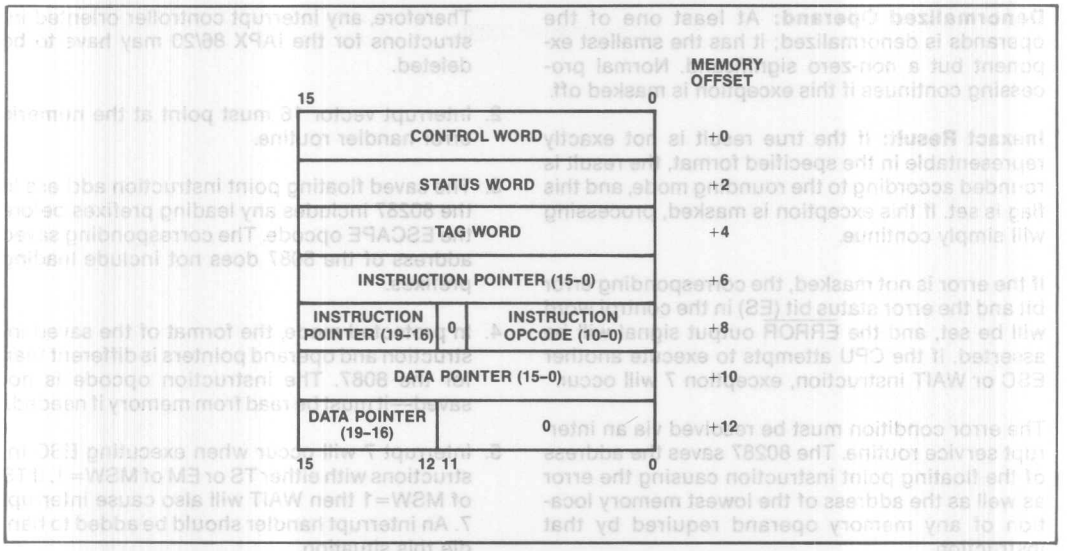


Figure 8b. Real Mode 80287 Instruction and Data Pointer Image in Memory

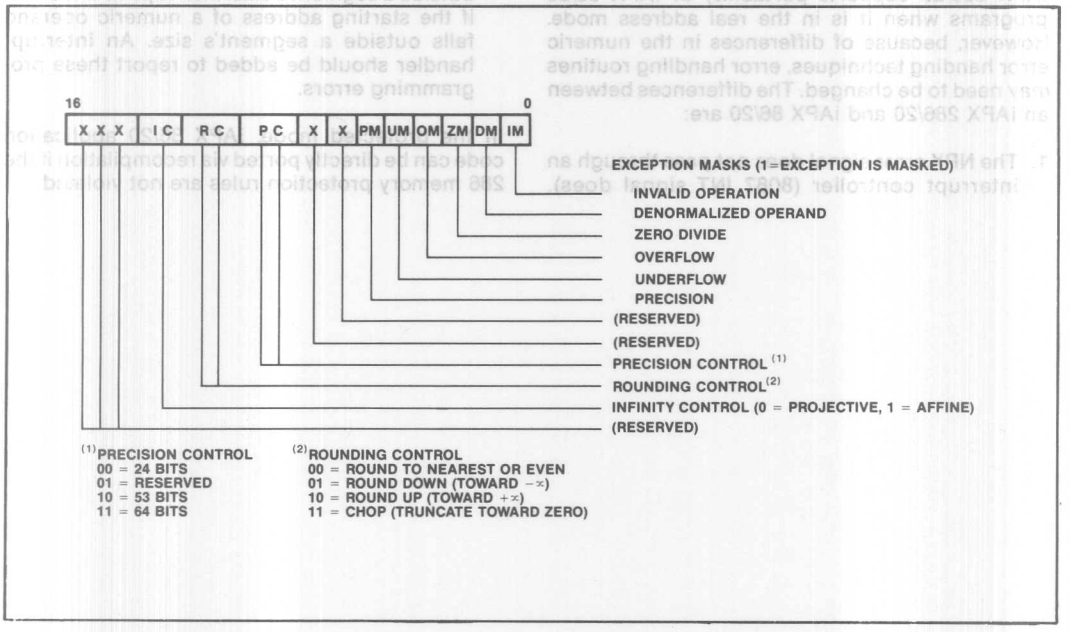


Figure 9. 80287 Control Word

210920-001

Denormalized Operand: At least one of the operands is denormalized; it has the smallest exponent but a non-zero significand. Normal processing continues if this exception is masked off.

Inexact Result: If the true result is not exactly representable in the specified format, the result is rounded according to the rounding mode, and this flag is set. If this exception is masked, processing will simply continue.

If the error is not masked, the corresponding error bit and the error status bit (ES) in the control word will be set, and the ERROR output signal will be asserted. If the CPU attempts to execute another ESC or WAIT instruction, exception 7 will occur.

The error condition must be resolved via an interrupt service routine. The 80287 saves the address of the floating point instruction causing the error as well as the address of the lowest memory location of any memory operand required by that instruction.

iAPX 86/20 COMPATIBILITY:

iAPX 286/20 supports portability of iAPX 86/20 programs when it is in the real address mode. However, because of differences in the numeric error handing techniques, error handling routines *may* need to be changed. The differences between an iAPX 286/20 and iAPX 86/20 are:

1. The NPX error signal does not pass through an interrupt controller (8087 INT signal does). Therefore, any interrupt controller oriented instructions for the iAPX 86/20 may have to be deleted.
2. Interrupt vector 16 must point at the numeric error handler routine.
3. The saved floating point instruction address in the 80287 includes any leading prefixes before the ESCAPE opcode. The corresponding saved address of the 8087 does not include leading prefixes.
4. In protected mode, the format of the saved instruction and operand pointers is different than for the 8087. The instruction opcode is not saved—it must be read from memory if needed.
5. Interrupt 7 will occur when executing ESC instructions with either TS or EM of MSW=1. If TS of MSW=1 then WAIT will also cause interrupt 7. An interrupt handler should be added to handle this situation.
6. Interrupt 9 will occur if the second or subsequent words of a floating point operand fall outside a segment's size. Interrupt 13 will occur if the starting address of a numeric operand falls outside a segment's size. An interrupt handler should be added to report these programming errors.

In the protected mode, iAPX 86/20 application code can be directly ported via recompilation if the 286 memory protection rules are not violated.

210920-001

ABSOLUTE MAXIMUM RATINGS*

Ambient Temperature Under Bias . . 0°C to 70°C
Storage Temperature −65°C to +150°C
Voltage on Any Pin with
Respect to Ground −1.0 to +7V
Power Dissipation 3.0 Watt

**NOTICE: Stresses above those listed under Absolute Maximum Ratings may cause permanent damage to the device. This is a stress rating only and functional operation of the device at these or any other conditions above those indicated in the operational sections of this specification is not implied. Exposure to absolute maximum rating conditions for extended periods may affect device reliability.*

D.C. CHARACTERISTICS $T_A = 0°C$ to $70°C$, $V_{CC} = 5V \pm 10\%$

Symbol	Parameter	Min.	Max.	Units	Test Conditions
V_{IL}	Input Low Voltage	−0.5	+0.8	V	
V_{IH}	Input High Voltage	2.0	V_{CC}+0.5	V	
V_{OL}	Output Low Voltage		0.45	V	I_{OL} = 3.0 mA
V_{OH}	Output High Voltage	2.4		V	I_{OH} = .400 μA
I_{CC}	Power Supply Current		475	mA	T_A = 25°C
I_{LI}	Input Leakage Current		±10	μA	$0V \leq V_{IN} \leq V_{CC}$
I_{LO}	Output Leakage Current		±10	μA	$0.45V \leq V_{OUT} \leq V_{CC}$
V_{CL}	Clock Input Low Voltage CKM=1 CKM=0	 −0.5 −0.5	 +0.6 +0.8	 V V	
V_{CH}	Clock Input High Voltage CKM=1 CKM=0	 3.8 2.0	 V_{CC}+1.0 V_{CC}+1.0	 V V	
C_{IN}	Capacitance of Input & Output Buffers (all except I/O Buffer and CLK)		10	pF	fc = 1 Mhz
C_{IO}	Capacitance of I/O Buffer for D_{15}–D_0		20	pF	fc = 1 MHz
C_{CLK}	Capacitance of CLK Input		12	pF	fc = 1 MHz

210920-001

A.C. CHARACTERISTICS T_A=0°C to 70°C, V_{CC} = 5V ± 10%
TIMING REQUIREMENTS

Symbol	Parameter	Min.	Max.	Units	Test Conditions
TCLCL	CLK cycle period			ns	
	CKM=1: 5MHz	200	500		
	8MHz	125	500		at 1.5 V
	CKM=0: 16MHz	62.5	250		
TCLCH	CLK low time CKM=1	118		ns	at 0.6 V
	CKM=0	15	230		at 0.8 V
TCHCL	CLK high time CKM=1	69		ns	at 3.8 V
	CKM=0	20	235		at 2.0 V
TCH1CH2	CLK rise time		10	ns	from 1.0 to 3.5 V
TCL2CL1	CLK fall time		10	ns	from 3.5 to 1.0V
TDVWH	Data valid set up to $\overline{NPWR}$ inactive	75		ns	
TWHDX	Data hold from $\overline{NPWR}$ inactive	0		ns	
TWLWH, TRLRH	$\overline{NPWR}$, $\overline{NPRD}$ active time	95		ns	
TAVRL, TAVWL	Command Valid to $\overline{NPWR}$ or $\overline{NPRD}$ active	0		ns	at 1.5 V
TMHRL	Minimum response from PEREQ active set up to $\overline{NPRD}$ active	130		ns	
TKLKH	$\overline{PEACK}$ active time	95		ns	
TKHKL	$\overline{PEACK}$ inactive time	95		ns	
TKHCH	$\overline{PEACK}$ inactive to $\overline{NPWR}$, $\overline{NPRD}$ inactive	0		ns	
TCHKL	Data Channel $\overline{NPRD}$, $\overline{NPWR}$ inactive set up to $\overline{PEACK}$ active	−25		ns	
TWHAX, TRHAX	Command hold from $\overline{NPWR}$, $\overline{NPRD}$ inactive	0		ns	
TKLCL	$\overline{PEACK}$ active set up to command active	0		ns	
T2CLCL	80286 Clock period	62.5	250	ns	
T2CLCH	80286 Clock low time	15	230	ns	at 0.8V
T2CHCL	80286 Clock high time	20	235	ns	at 2.0V

210920-001

A.C. CHARACTERISTICS T_A=0°C to 70°C, V_{CC} = 5V± 10%
TIMING REQUIREMENTS (cont.)

Symbol	Parameter	Min.	Max.	Units	Test Conditions
TCLSH	$\overline{S1}$, $\overline{S0}$ hold time	0		ns	at 1.5 V
TSVCL	$\overline{S1}$, $\overline{S0}$ valid setup time	22.5		ns	
TCIVCL	COD/$\overline{INTA}$ valid setup time	0		ns	
TCLCIH	COD/$\overline{INTA}$ hold time	0		ns	
TRVCL	$\overline{READY}$ valid setup time	38.5		ns	
TCLRH	$\overline{READY}$ hold time	25		ns	
THVCL	HLDA valid setup time	0		ns	
TCLHH	HLDA hold time	0		ns	
TCLIH	Input from CLK hold time	45		ns	at 1.5V (See Note 1)
TIVCH	Input to CLK setup time	70		ns	

NOTE:
1. To guarantee a predetermined response for testing purposes.

A.C. CHARACTERISTICS—TIMING RESPONSES

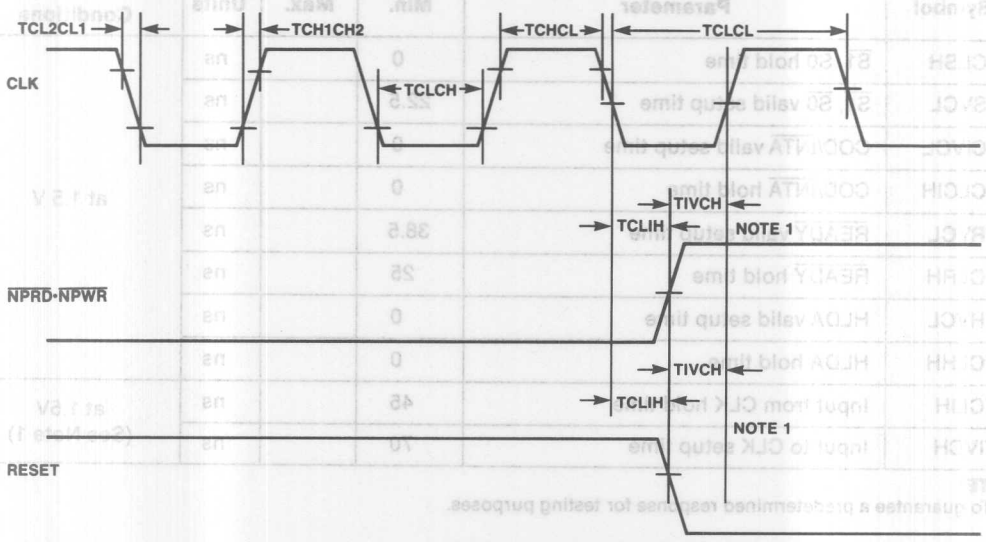
Symbol	Parameter	Min.	Max.	Units	Test Conditions
TRHQZ	$\overline{NPRD}$ inactive to data tri-state		37.5	ns	CL=20pF–100pF at 1.5V
TRLQV	$\overline{NPRD}$ active to data valid		60	ns	
TILBH	$\overline{ERROR}$ active to $\overline{BUSY}$ inactive	100		ns	CL=100pF at 1.5V
TWLBV	$\overline{NPWR}$ active to $\overline{BUSY}$ valid		100	ns	
TCLML	$\overline{NPRD}$, $\overline{NPWR}$ active to PEREQ inactive		120	ns	CL = 100pF at 1.5V (See Note 1)
TKLML	$\overline{PEACK}$ active to PEREQ inactive		127	ns	
TCMDI	Minimum command inactive time				at 1.5V
	Write-Write	95		ns	
	Read-Read	150		ns	
	Write-Read	105		ns	
	Read-Write	95		ns	
TRHQH	Data valid hold from $\overline{NPRD}$ inactive	5		ns	CL=40pF at 1.5V

NOTE:
1. On last data transfer of numeric instruction.

210920-001

WAVEFORMS

CLOCK TIMING

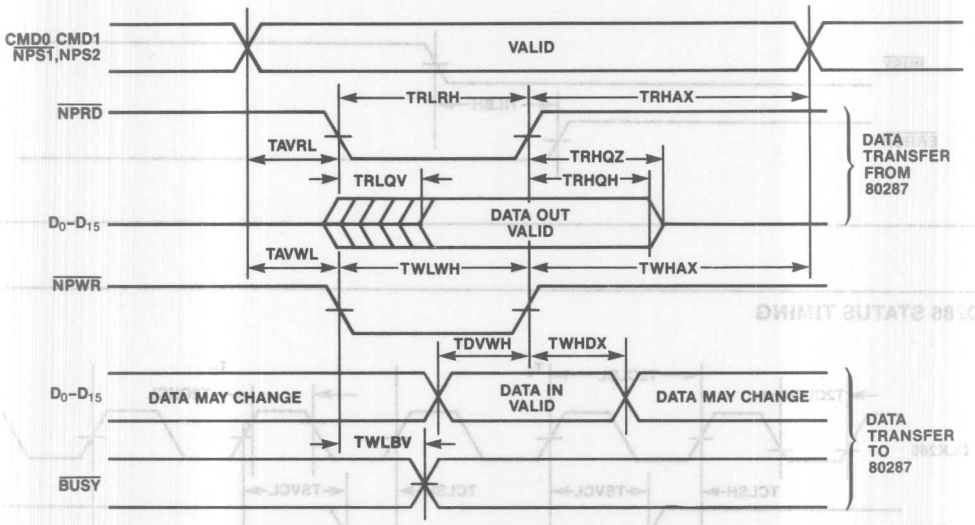


NOTE:
1. Shown for CKM=1, when CKM=0 TIVCH is measured from the second falling edge of CLK. These inputs may be asynchronous to CLK. This input timing guarantees a predetermined response.

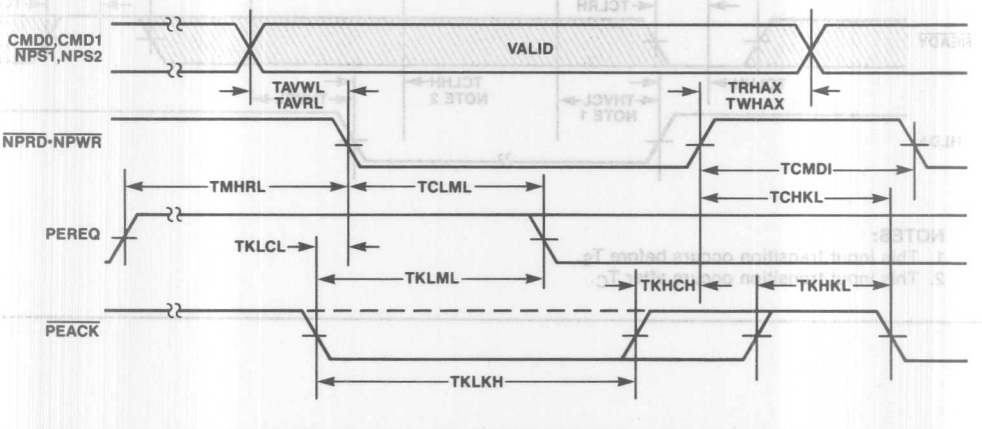
210920-001

WAVEFORMS (cont.)

DATA TRANSFER TIMING



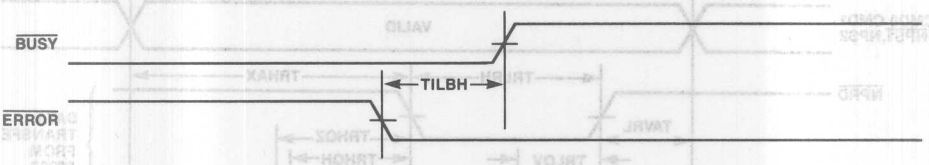
DATA CHANNEL TIMING



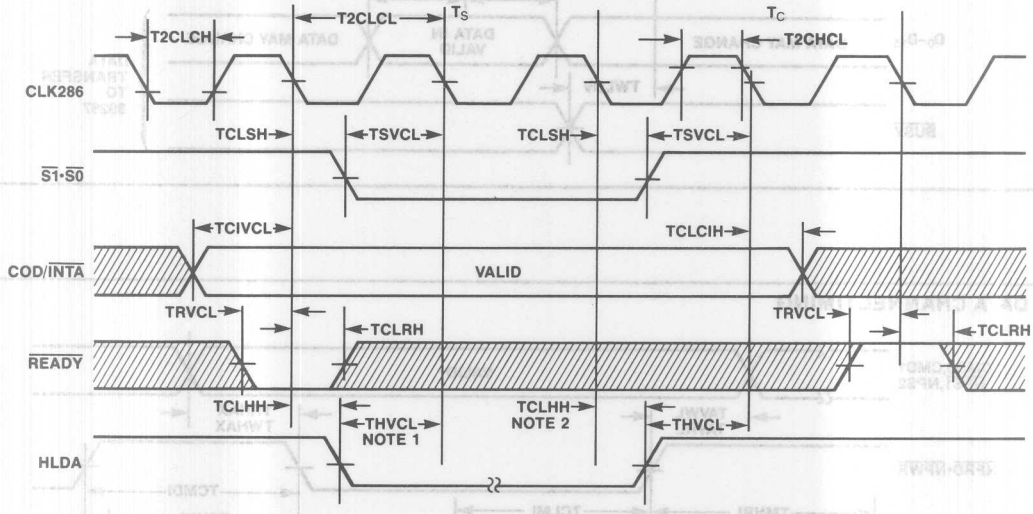
210920-001

WAVEFORMS (cont.)

ERROR OUTPUT TIMING



80286 STATUS TIMING



NOTES:
1. This input transition occurs before T_S.
2. This input transition occurs after T_C.

210920-001

Table 6. 80287 Extensions to the 80286 Instruction Set

Data Transfer		Optional 8,16 Bit Displacement	Clock Count Range 32 Bit Real	32 Bit Integer	64 Bit Real	16 Bit Integer
FLD = LOAD	MF =		00	01	10	11
Integer/Real Memory to ST(0)	ESCAPE MF 1 \| MOD 0 0 0 R/M	DISP	38–56	52–60	40–60	46–54
Long Integer Memory to ST(0)	ESCAPE 1 1 1 \| MOD 1 0 1 R/M	DISP	60–68			
Temporary Real Memory to ST(0)	ESCAPE 0 1 1 \| MOD 1 0 1 R/M	DISP	53–65			
BCD Memory to ST(0)	ESCAPE 1 1 1 \| MOD 1 0 0 R/M	DISP	290–310			
ST(i) to ST(0)	ESCAPE 0 0 1 \| 1 1 0 0 0 ST(i)		17–22			
FST = STORE						
ST(0) to Integer/Real Memory	ESCAPE MF 1 \| MOD 0 1 0 R/M	DISP	84–90	82–92	96–104	80–90
ST(0) to ST(i)	ESCAPE 1 0 1 \| 1 1 0 1 0 ST(i)		15–22			
FSTP = STORE AND POP						
ST(0) to Integer/Real Memory	ESCAPE MF 1 \| MOD 0 1 1 R/M	DISP	86–92	84–94	98–106	82–92
ST(0) to Long Integer Memory	ESCAPE 1 1 1 \| MOD 1 1 1 R/M	DISP	94–105			
ST(0) to Temporary Real Memory	ESCAPE 0 1 1 \| MOD 1 1 1 R/M	DISP	52–58			
ST(0) to BCD Memory	ESCAPE 1 1 1 \| MOD 1 1 0 R/M	DISP	520–540			
ST(0) to ST(i)	ESCAPE 1 0 1 \| 1 1 0 1 1 ST(i)		17–24			
FXCH = Exchange ST(i) and ST(0)	ESCAPE 0 0 1 \| 1 1 0 0 1 ST(i)		10–15			
Comparison						
FCOM = Compare						
Integer/Real Memory to ST(0)	ESCAPE MF 0 \| MOD 0 1 0 R/M	DISP	60–70	78–91	65–75	72–86
ST(i) to ST (0)	ESCAPE 0 0 0 \| 1 1 0 1 0 ST(i)		40–50			
FCOMP = Compare and Pop						
Integer/Real Memory to ST(0)	ESCAPE MF 0 \| MOD 0 1 1 R/M	DISP	63–73	80–93	67–77	74–88
ST(i) to ST(0)	ESCAPE 0 0 0 \| 1 1 0 1 1 ST(i)		45–52			
FCOMPP = Compare ST(1) to ST(0) and Pop Twice	ESCAPE 1 1 0 \| 1 1 0 1 1 0 0 1		45–55			
FTST = Test ST(0)	ESCAPE 0 0 1 \| 1 1 1 0 0 1 0 0		38–48			
FXAM = Examine ST(0)	ESCAPE 0 0 1 \| 1 1 1 0 0 1 0 1		12–23			

Mnemonics © Intel 1982

210920-001

Table 6. 80287 Extensions to the 80286 Instruction Set (cont.)

Constants		Optional 8,16 Bit Displacement	Clock Count Range			
			32 Bit Real	32 Bit Integer	64 Bit Real	16 Bit Integer
	MF =		00	01	10	11
FLDZ = LOAD + 0.0 into ST(0)	ESCAPE 0 0 1 \| 1 1 1 0 1 1 1 0		11–17			
FLD1 = LOAD + 1.0 into ST(0)	ESCAPE 0 0 1 \| 1 1 1 0 1 0 0 0		15–21			
FLDPI = LOAD π into ST(0)	ESCAPE 0 0 1 \| 1 1 1 0 1 0 1 1		16–22			
FLDL2T = LOAD $\log_2 10$ into ST(0)	ESCAPE 0 0 1 \| 1 1 1 0 1 0 0 1		16–22			
FLDL2E = LOAD $\log_2 e$ into ST(0)	ESCAPE 0 0 1 \| 1 1 1 0 1 0 1 0		15–21			
FLDLG2 = LOAD $\log_{10} 2$ into ST(0)	ESCAPE 0 0 1 \| 1 1 1 0 1 1 0 0		18–24			
FLDLN2 = LOAD $\log_e 2$ into ST(0)	ESCAPE 0 0 1 \| 1 1 1 0 1 1 0 1		17–23			
Arithmetic						
FADD = Addition						
Integer/Real Memory with ST(0)	ESCAPE MF 0 \| MOD 0 0 0 R/M	DISP	90–120	108–143	95–125	102–137
ST(i) and ST(0)	ESCAPE d P 0 \| 1 1 0 0 0 ST(i)		70–100 (Note 1)			
FSUB = Subtraction						
Integer/Real Memory with ST(0)	ESCAPE MF 0 \| MOD 1 0 R R/M	DISP	90–120	108–143	95–125	102–137
ST(i) and ST(0)	ESCAPE d P 0 \| 1 1 1 0 R R/M		70–100 (Note 1)			
FMUL = Multiplication						
Integer/Real Memory with ST(0)	ESCAPE MF 0 \| MOD 0 0 1 R/M	DISP	110–125	130–144	112–168	124–138
ST(i) and ST(0)	ESCAPE d P 0 \| 1 1 0 0 1 R/M		90–145 (Note 1)			
FDIV = Division						
Integer/Real Memory with ST(0)	ESCAPE MF 0 \| MOD 1 1 R R/M	DISP	215–225	230–243	220–230	224–238
ST(i) and ST(0)	ESCAPE d P 0 \| 1 1 1 1 R R/M		193–203 (Note 1)			
FSQRT = Square Root of ST(0)	ESCAPE 0 0 1 \| 1 1 1 1 1 0 1 0		180–186			
FSCALE = Scale ST(0) by ST(1)	ESCAPE 0 0 1 \| 1 1 1 1 1 1 0 1		32–38			
FPREM = Partial Remainder of ST(0) ÷ ST(1)	ESCAPE 0 0 1 \| 1 1 1 1 1 0 0 0		15–190			
FRNDINT = Round ST(0) to Integer	ESCAPE 0 0 1 \| 1 1 1 1 1 1 0 0		16–50			

NOTE.
1. If P=1 then add 5 clocks.

210920-001

Table 6. 80287 Extensions to the 80286 Instruction Set (cont.)

		Optional 8,16 Bit Displacement	Clock Count Range
FXTRACT = Extract Components of St(0)	ESCAPE 0 0 1 \| 1 1 1 1 0 1 0 0		27–55
FABS = Absolute Value of ST(0)	ESCAPE 0 0 1 \| 1 1 1 0 0 0 0 1		10–17
FCHS = Change Sign of ST(0)	ESCAPE 0 0 1 \| 1 1 1 0 0 0 0 0		10–17
Transcendental			
FPTAN = Partial Tangent of ST(0)	ESCAPE 0 0 1 \| 1 1 1 1 0 0 1 0		30–540
FPATAN = Partial Arctangent of ST(0) ÷ST(1)	ESCAPE 0 0 1 \| 1 1 1 1 0 0 1 1		250–800
F2XM1 = $2^{ST(0)}-1$	ESCAPE 0 0 1 \| 1 1 1 1 0 0 0 0		310–630
FYL2X = ST(1) • Log_2 [ST(0)]	ESCAPE 0 0 1 \| 1 1 1 1 0 0 0 1		900–1100
FYL2XP1 = ST(1) • Log_2 [ST(0) +1[	ESCAPE 0 0 1 \| 1 1 1 1 1 0 0 1		700–1000
Processor Control			
FINIT = Initialize NPX	ESCAPE 0 1 1 \| 1 1 1 0 0 0 1 1		2–8
FSETPM = Enter Protected Mode	ESCAPE 0 1 1 \| 1 1 1 0 0 1 0 0		2–8
FSTSW AX = Store Control Word	ESCAPE 1 1 1 \| 1 1 1 0 0 0 0 0		10–16
FLDCW = Load Control Word	ESCAPE 0 0 1 \| MOD 1 0 1 R/M	DISP	7–14
FSTCW = Store Control Word	ESCAPE 0 0 1 \| MOD 1 1 1 R/M	DISP	12–18
FSTSW = Store Status Word	ESCAPE 1 0 1 \| MOD 1 1 1 R/M	DISP	12–18
FCLEX = Clear Exceptions	ESCAPE 0 1 1 \| 1 1 1 0 0 0 1 0		2–8
FSTENV = Store Environment	ESCAPE 0 0 1 \| MOD 1 1 0 R/M	DISP	40–50
FLDENV = Load Environment	ESCAPE 0 0 1 \| MOD 1 0 0 R/M	DISP	35–45
FSAVE = Save State	ESCAPE 1 0 1 \| MOD 1 1 0 R/M	DISP	205–215
FRSTOR = Restore State	ESCAPE 1 0 1 \| MOD 1 0 0 R/M	DISP	205–215
FINCSTP = Increment Stack Pointer	ESCAPE 0 0 1 \| 1 1 1 1 0 1 1 1		6–12
FDECSTP = Decrement Stack Pointer	ESCAPE 0 0 1 \| 1 1 1 1 0 1 1 0		6–12

210920-001

Table 6. 80287 Extensions to the 80286 Instruction Set (cont.)

		Clock Count
FFREE = Free ST(i)	ESCAPE 1 0 1 \| 1 1 0 0 0 ST(i)	9–16
FNOP = No Operation	ESCAPE 0 0 1 \| 1 1 0 1 0 0 0 0	10–16

NOTES:

1. if mod=00 then DISP=0*, disp-low and disp-high are absent
 if mod=01 then DISP=disp-low sign-extended to 16-bits, disp-high is absent
 if mod=10 then DISP=disp-high; disp-low
 if mod=11 then r/m is treated as an ST(i) field
2. if r/m=000 then EA=(BX) + (SI) +DISP
 if r/m=001 then EA=(BX) + (DI) +DISP
 if r/m=010 then EA=(BP) + (SI) +DISP
 if r/m=011 then EA=(BP) + (DI) +DISP
 if r/m=100 then EA=(SI) + DISP
 if r/m=101 then EA=(DI) + DISP
 if r/m=110 then EA=(BP) + DISP
 if r/m=111 then EA=(BX) + DISP
 *except if mod=000 and r/m=110 then EA =disp-high; disp-low.
3. MF= Memory Format
 00—32-bit Real
 01—32-bit Integer
 10—64-bit Real
 11—16-bit Integer
4. ST(0)= Current stack top
 ST(i) i^{th} register below stack top
5. d= Destination
 0—Destination is ST(0)
 1—Destination is ST(i)
6. P= Pop
 0—No pop
 1—Pop ST(0)
7. R= Reverse: When d=1 reverse the sense of R
 0—Destination (op) Source
 1—Source (op) Destination
8. For **FSQRT:** $-0 \leq ST(0) \leq +\infty$
 For **FSCALE:** $-2^{15} \leq ST(1) < +2^{15}$ and ST(1) integer
 For **F2XM1:** $0 \leq ST(0) \leq 2^{-1}$
 For **FYL2X:** $0 < ST(0) < \infty$
 $-\infty < ST(1) < +\infty$
 For **FYL2XP1:** $0 \leq |ST(0)| < (2 - \sqrt{2})/2$
 $-\infty < ST(1) < \infty$
 For **FPTAN:** $0 \leq ST(0) \leq \pi/4$
 For **FPATAN:** $0 \leq ST(0) < ST(1) < +\infty$

82284
CLOCK GENERATOR AND READY INTERFACE FOR iAPX 286 PROCESSORS

- **Generates System Clock for iAPX 286 Processors**
- **Uses Crystal or TTL Signal for Frequency Source**
- **Provides Local READY and Multibus* READY Synchronization**
- **18-pin Package**
- **Single +5V Power Supply**
- **Generates System Reset Output from Schmitt Trigger Input**
- **Available in EXPRESS**
 - **Standard Temperature Range**
 - **Extended Temperature Range**

The 82284 is a clock generator/driver which provides clock signals for iAPX 286 processors and support components. It also contains logic to supply $\overline{\text{READY}}$ to the CPU from either asynchronous or synchronous sources and synchronous RESET from an asynchronous input with hysteresis.

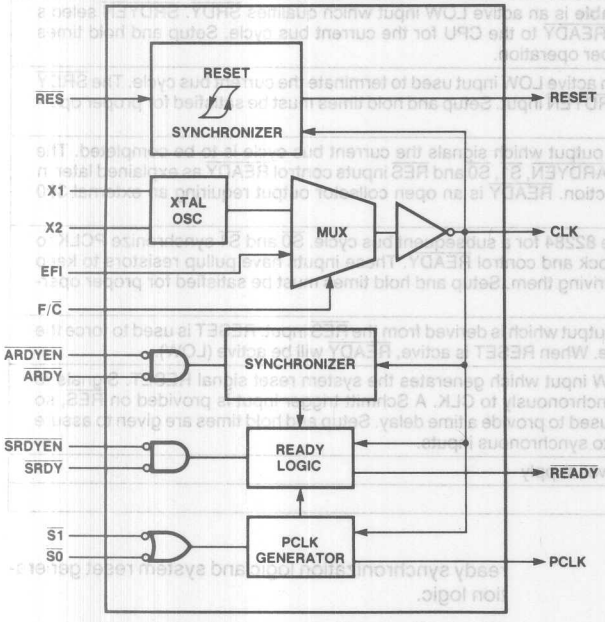


Figure 1. 82284 Block Diagram

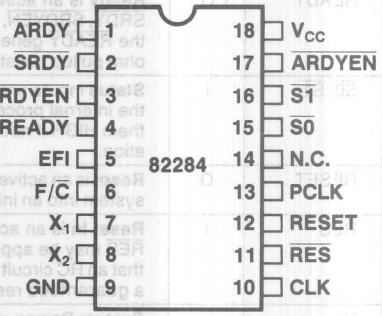


Figure 2. 82284 Pin Configuration

* Multibus is a patented bus of Intel

Intel Corporation Assumes No Responsibility for the Use of Any Circuitry Other Than Circuitry Embodied in an Intel Product. No Other Circuit Patent Licenses are implied.

© INTEL CORPORATION 1982

MAY 1982
ORDER NUMBER: 210453-001

Table 1. Pin Description

The following pin function descriptions are for the 82284 clock generator.

Symbol	Type	Name and Function
CLK	O	**System Clock** is the signal used by the processor and support devices which must be synchronous with the processor. The frequency of the CLK output has twice the desired internal processor clock frequency. CLK can drive both TTL and MOS level inputs.
F/$\overline{C}$	I	**Frequency/Crystal Select** is a strapping option to select the source for the CLK output. When F/$\overline{C}$ is strapped LOW, the internal crystal oscillator drives CLK. When F/$\overline{C}$ is strapped HIGH, the EFI input drives the CLK output.
X1, X2	I	**Crystal In** are the pins to which a parallel resonant fundamental mode crystal is attached for the internal oscillator. When F/$\overline{C}$ is LOW, the internal oscillator will drive the CLK output at the crystal frequency. The crystal frequency must be twice the desired internal processor clock frequency.
EFI	I	**External Frequency In** drives CLK when the F/$\overline{C}$ input is strapped HIGH. The EFI input frequency must be twice the desired internal processor clock frequency.
PCLK	O	**Peripheral Clock** is an output which provides a 50% duty cycle clock with 1/2 the frequency of CLK. PLCK will be in phase with the internal processor clock following the first bus cycle after the processor has been reset.
$\overline{ARDYEN}$	I	**Asynchronous Ready Enable** is an active LOW input which qualifies the $\overline{ARDY}$ input. $\overline{ARDYEN}$ selects $\overline{ARDY}$ as the source of ready for the current bus cycle. Inputs to $\overline{ARDYEN}$ may be applied asynchronously to CLK. Setup and hold times are given to assure a guaranteed response to synchronous inputs.
$\overline{ARDY}$	I	**Asynchronous Ready** is an active LOW input used to terminate the current bus cycle. The $\overline{ARDY}$ input is qualified by $\overline{ARDYEN}$. Inputs to $\overline{ARDY}$ may be applied asynchronously to CLK. Setup and hold times are given to assure a guaranteed response to synchronous inputs.
$\overline{SRDYEN}$	I	**Synchronous Ready Enable** is an active LOW input which qualifies $\overline{SRDY}$. $\overline{SRDYEN}$ selects $\overline{SRDY}$ as the source for $\overline{READY}$ to the CPU for the current bus cycle. Setup and hold times must be satisfied for proper operation.
$\overline{SRDY}$	I	**Synchronous Ready** is an active LOW input used to terminate the current bus cycle. The $\overline{SRDY}$ input is qualified by the $\overline{SRDYEN}$ input. Setup and hold times must be satisfied for proper operation.
$\overline{READY}$	O	**Ready** is an active LOW output which signals the current bus cycle is to be completed. The $\overline{SRDY}$, $\overline{SRDYEN}$, $\overline{ARDY}$, $\overline{ARDYEN}$, $\overline{S1}$, $\overline{S0}$ and $\overline{RES}$ inputs control $\overline{READY}$ as explained later in the $\overline{READY}$ generator section. $\overline{READY}$ is an open collector output requiring an external 300 ohm pullup resistor.
$\overline{S0}$, $\overline{S1}$	I	**Status** inputs prepare the 82284 for a subsequent bus cycle. $\overline{S0}$ and $\overline{S1}$ synchronize PCLK to the internal processor clock and control $\overline{READY}$. These inputs have pullup resistors to keep them HIGH if nothing is driving them. Setup and hold times must be satisfied for proper operation.
RESET	O	**Reset** is an active HIGH output which is derived from the $\overline{RES}$ input. RESET is used to force the system into an initial state. When RESET is active, $\overline{READY}$ will be active (LOW).
$\overline{RES}$	I	**Reset In** is an active LOW input which generates the system reset signal RESET. Signals to $\overline{RES}$ may be applied asynchronously to CLK. A Schmitt trigger input is provided on $\overline{RES}$, so that an RC circuit can be used to provide a time delay. Setup and hold times are given to assure a guaranteed response to synchronous inputs.
V_{CC}		**System Power**: +5V power supply
GND		**System Ground**: 0 volts

FUNCTIONAL DESCRIPTION

Introduction

The 82284 generates the clock, ready, and reset signals required for iAPX 286 processors and support components. The 82284 is packaged in an 18-pin DIP and contains a crystal controlled oscillator, MOS clock generator, peripheral clock generator, Multibus ready synchronization logic and system reset generation logic.

Clock Generator

The CLK output provides the basic timing control for an iAPX 286 system. CLK has output characteristics sufficient to drive MOS devices. CLK is generated by either an internal crystal oscillator or an external source as selected by the F/$\overline{C}$ strapping option. When

AFN-00786B

F/$\overline{C}$ is LOW, the crystal oscillator drives the CLK output. When F/$\overline{C}$ is HIGH, the EFI input drives the CLK output.

The 82284 provides a second clock output (PCLK) for peripheral devices. PCLK is CLK divided by two. PCLK has a duty cycle of 50% and TTL output drive characteristics. PCLK is normally synchronized to the internal processor clock.

After reset, the PCLK signal may be out of phase with the internal processor clock. The $\overline{S1}$ and $\overline{S0}$ signals of the first bus cycle are used to synchronize PCLK to the internal processor clock. The phase of the PCLK output changes by extending its HIGH time beyond one system clock (see waveforms). PCLK is forced HIGH whenever either $\overline{S0}$ or $\overline{S1}$ were active (LOW) for the two previous CLK cycles. PCLK continues to oscillate when both $\overline{S0}$ and $\overline{S1}$ are HIGH.

Since the phase of the internal processor clock will not change except during reset, the phase of PCLK will not change except during the first bus cycle after reset.

Oscillator

The oscillator circuit of the 82284 is a linear Pierce oscillator which requires an external parallel resonant, fundamental mode, crystal. The output of the oscillator is internally buffered. The crystal frequency chosen should be twice the required internal processor clock frequency. The crystal should have a typical load capacitance of 32 pF.

X1 and X2 are the oscillator crystal connections. For stable operation of the oscillator, two loading capacitors are recommended, as shown in Figure 3. The sum of the board capacitance and loading capacitance should equal the values shown. It is advisable to limit stray board capacitances (not including the effect of the loading capacitors or crystal capacitance) to less than 10 pF between the X1 and X2 pins.

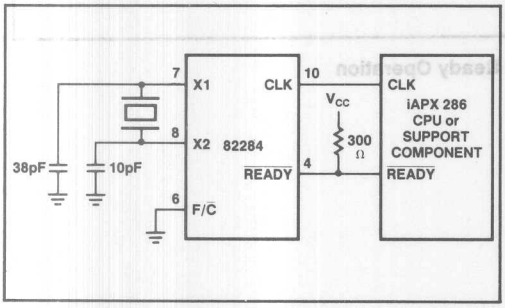


Figure 3. Recommended Crystal and Ready Connections

Reset Operation

The reset logic provides the RESET output to force the system into a known, initial state. When the $\overline{RES}$ input is active (LOW), the RESET output becomes active (HIGH). $\overline{RES}$ is synchronized internally at the falling edge of CLK before generating the RESET output (see waveforms). Synchronization of the $\overline{RES}$ input introduces a one or two CLK delay before affecting the RESET output.

At power up, a system does not have a stable V_{CC} and CLK. To prevent spurious activity, $\overline{RES}$ should be asserted until V_{CC} and CLK stabilize at their operating values. iAPX 286 processors and support components also require their RESET inputs be HIGH a minimum number of CLK cycles. An RC network, as shown in Figure 4, will keep $\overline{RES}$ LOW long enough to satisfy both needs.

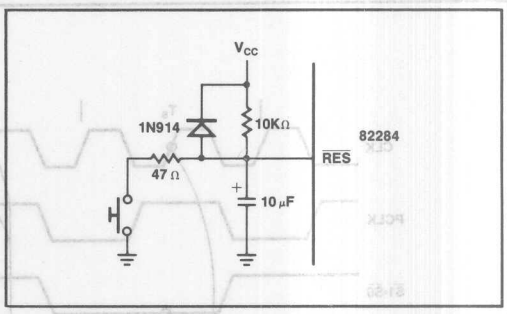


Figure 4. Typical RC RESET Timing Circuit

A Schmitt trigger input with hysteresis on $\overline{RES}$ assures a single transition of RESET with an RC circuit on $\overline{RES}$. The hysteresis separates the input voltage level at which the circuit output switches between HIGH to LOW from the input voltage level at which the circuit output switches between LOW to HIGH. The $\overline{RES}$ HIGH to LOW input transition voltage is lower than the $\overline{RES}$ LOW to HIGH input transition voltage. As long as the slope of the $\overline{RES}$ input voltage remains in the same direction (increasing or decreasing) around the $\overline{RES}$ input transition voltage, the RESET output will make a single transition.

Ready Operation

The 82284 accepts two ready sources for the system ready signal which terminates the current bus cycle. Either a synchronous ($\overline{SRDY}$) or asynchronous ready ($\overline{ARDY}$) source may be used. Each ready input has an enable ($\overline{SRDYEN}$ and $\overline{ARDYEN}$) for selecting the type of ready source required to terminate the current bus cycle. An address decoder would normally select one of the enable inputs.

AFN-00786B

is enabled (LOW), if either SRDY + SRDYEN = 0 or ARDY + ARDYEN = 0 when sampled by the 82284 READY generation logic. READY will remain active for at least two CLK cycles.

The READY output has an open-collector driver allowing other ready circuits to be wire or'ed with it. The READY signal of an iAPX 286 system requires an external 300 ohm pull-up resistor. To force the READY signal inactive (HIGH) at the start of a bus cycle, the READY output floats when either S1 or S0 are sampled LOW at the falling edge of CLK. Two system clock periods are allowed for the pull-up resistor to pull the READY signal to V_{IH}. When RESET is active, READY is forced active one CLK later (see waveforms).

Figure 5 illustrates the operation of SRDY and SRDYEN. These inputs are sampled on the falling edge of CLK when S1 and S0 are inactive and PCLK is HIGH. READY is forced active when both SRDY and SRDYEN are sampled as LOW.

Figure 6 shows the operation of ARDY and ARDYEN. These inputs are sampled by an internal synchronizer at each falling edge of CLK. The output of the synchronizer is then sampled when PCLK is HIGH. If the synchronizer resolved both the ARDY and ARDYEN inputs to have been LOW, READY becomes LOW. When both ARDY and ARDYEN have been resolved as active, the SRDY and SRDYEN inputs are ignored.

READY remains active until either S1 or S0 are sampled LOW, or the ready inputs are sampled as inactive.

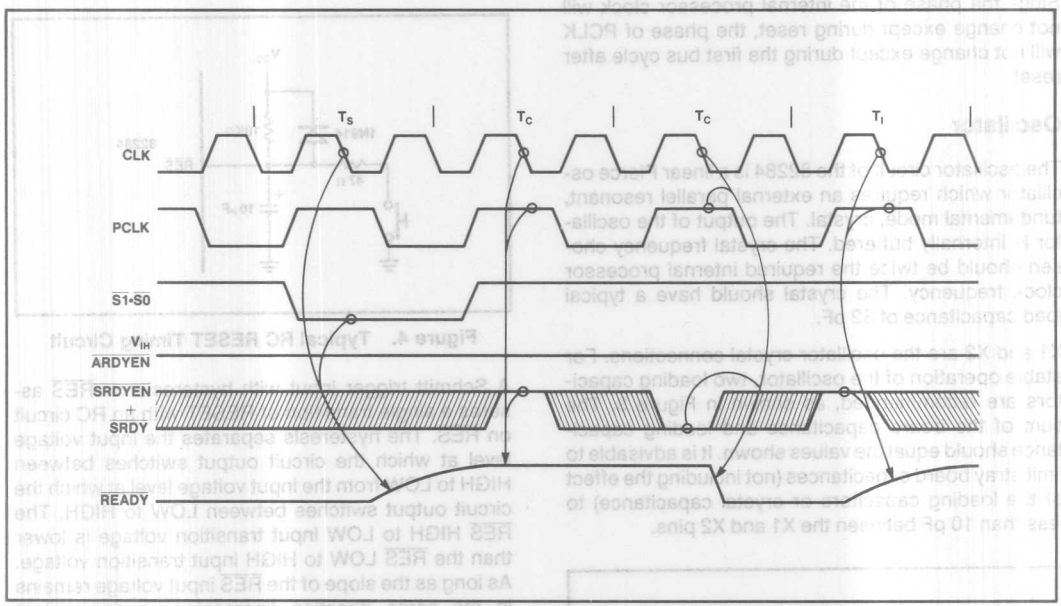


Figure 5. Synchronous Ready Operation

AFN-00786B

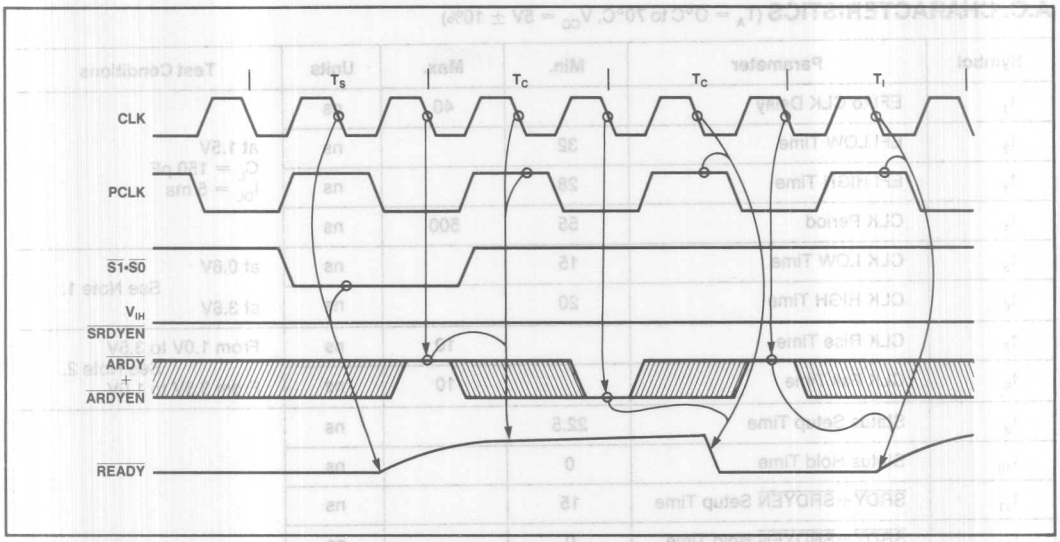


Figure 6. Asynchronous Ready Operation

ABSOLUTE MAXIMUM RATINGS*

Temperature Under Bias 0°C to 70°C
Storage Temperature -65°C to +150°C
All Output and Supply Voltages -0.5V to +7V
All Input Voltages -1.0V to +5.5V
Power Dissipation 1 Watt

**Notice: Stresses above those listed under "Absolute Maxmum Ratings" may cause permanent damage to the device. This is a stress rating only and functional operation of the device at these or any other conditions above those indicated in the operational sections of this specification is not implied. Exposure to absolute maximum rating conditions for extended periods may affect device reliability.*

D.C. CHARACTERISTICS (T_A = 0°C to 70°C, V_{CC} = 5V ± 10%)

Symbol	Parameter	Min.	Max.	Units	Test Conditions
I_F	Forward Input Current		−0.5	mA	V_F = 0.45V
I_R	Reverse Input Current		50	uA	V_R = 5.25V
V_C	Input Forward Clamp Voltage		-1.0	V	I_C = -5 mA
I_{CC}	Power Supply Current		145	mA	
V_{IL}	Input LOW Voltage		0.8	V	
V_{IH}	Input HIGH Voltage	2.0		V	
V_{OL},V_{CL}	Output LOW Voltage		0.45	V	I_{OL} = 5 mA
V_{CH}	CLK Output HIGH Voltage	4.0		V	I_{OH} = -1 mA
V_{OH}	Output HIGH Voltage	2.4		V	I_{OH} = -1 mA
V_{IH_R}	$\overline{RES}$ Input HIGH Voltage	2.6		V	
V_{IH_R}-V_{IL_R}	$\overline{RES}$ Input Hysteresis	0.25		V	
C_I	Input Capacitance		10	pF	

AFN-00786B

A.C. CHARACTERISTICS (T_A = O°C to 70°C, V_{CC} = 5V ± 10%)

Symbol	Parameter	Min.	Max.	Units	Test Conditions
t_1	EFI to CLK Delay		40	ns	at 1.5V C_L = 150 pF I_{OL} = 5 ma
t_2	EFI LOW Time	32		ns	
t_3	EFI HIGH Time	28		ns	
t_4	CLK Period	55	500	ns	
t_5	CLK LOW Time	15		ns	at 0.6V See Note 1.
t_6	CLK HIGH Time	20		ns	at 3.8V
t_7	CLK Rise Time		10	ns	From 1.0V to 3.5V See Note 2.
t_8	CLK Fall Time		10	ns	From 3.5V to 1.0V
t_9	Status Setup Time	22.5		ns	at 0.8V and 2.0V on input and 0.8V on CLK
t_{10}	Status Hold Time	0		ns	
t_{11}	$\overline{SRDY}$+$\overline{SRDYEN}$ Setup Time	15		ns	
t_{12}	$\overline{SRDY}$+$\overline{SRDYEN}$ Hold Time	0		ns	
t_{13}	$\overline{ARDY}$+$\overline{ARDYEN}$ Setup Time	0		ns	
t_{14}	$\overline{ARDY}$+$\overline{ARDYEN}$ Hold Time	16		ns	
t_{15}	$\overline{RES}$ Setup Time	16		ns	
t_{16}	$\overline{RES}$ Hold Time	0		ns	
t_{17}	$\overline{READY}$ Inactive Delay	5		ns	at 0.8V C_L = 150 pF I_{OL} = 20 mA
t_{18}	$\overline{READY}$ Active Delay	0	24	ns	
t_{19}	PCLK Delay	0	40	ns	at 0.8V on CLK to See Note 3.
t_{20}	RESET Delay	0	40	ns	0.8V or 2.0V on output
t_{21}	PCLK Low Time	t_4-12.5		ns	at 0.6V See Note 3.
t_{22}	PCLK High Time	t_4-12.5		ns	at 3.8V

Note 1. C_L = 150 pF, I_{OL} = 5 ma. With either the internal oscillator with the recommended crystal and load or the EFI input.

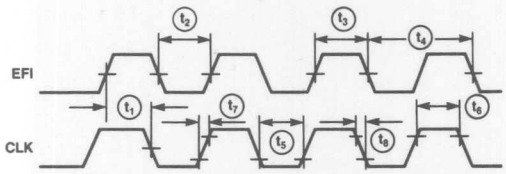
Note 2. C_L = 150 pF
I_{OL} = 5 mA

Note 3. I_{OL} = 5 mA
C_L = 75 pF

AFN-00786B

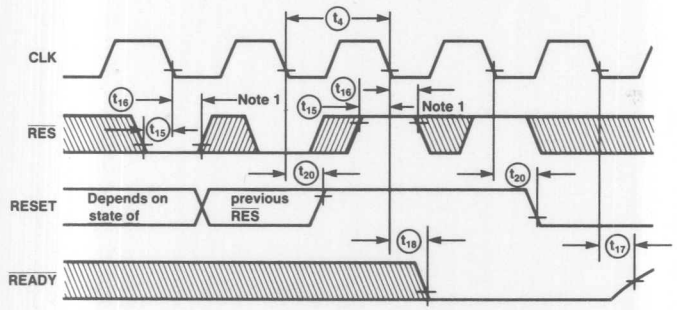
Waveforms

CLK as a Function of EFI



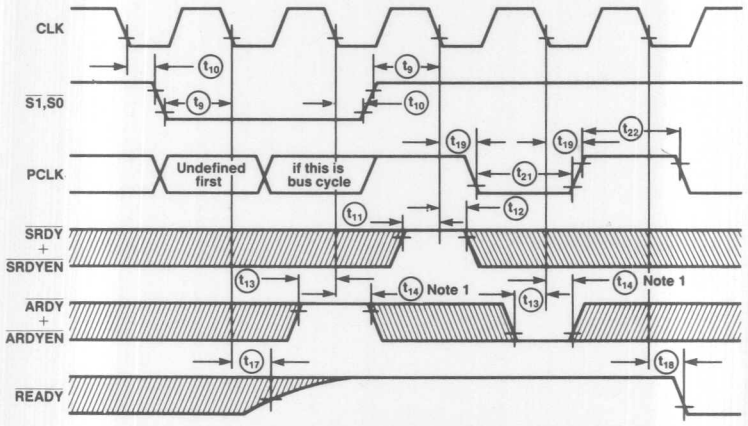
NOTE: The EFI input LOW and HIGH times as shown are required to guarentee the CLK LOW and HIGH times shown.

RESET and $\overline{\text{READY}}$ Timing as a Function of $\overline{\text{RES}}$ with $\overline{\text{S1}}$ and $\overline{\text{S0}}$ HIGH



NOTE 1: This is an asynchronous input. The setup and hold times shown are required to guarantee the response shown.

$\overline{\text{READY}}$ and PCLK Timing with $\overline{\text{RES}}$ HIGH



NOTE 1: This is an asynchronous input. The setup and hold times shown are required to guarantee the response shown.

AFN-00786B

CLK as a Function of EFI

NOTE: The EFI input LOW and HIGH times as shown are required to guarantee the CLK LOW and HIGH times shown.

RESET and READY Timing as a Function of RES with S1 and S0 HIGH

NOTE 1: This is an asynchronous input. The setup and hold times shown are required to guarantee the response shown.

READY and PCLK Timing with RES HIGH

NOTE 1: This is an asynchronous input. The setup and hold times shown are required to guarantee the response shown.

82288
BUS CONTROLLER
FOR iAPX 286 PROCESSORS

- **Provides Commands and Control for Local and System Bus**
- **Offers Wide Flexibility in System Configurations**
- **Flexible Command Timing**
- **Optional Multibus* Compatible Timing**
- **Control Drivers with 16 ma I_{OL} and 3-State Command Drivers with 32 ma I_{OL}**
- **Single +5V Supply**

The Intel 82288 Bus Controller is a 20-pin HMOS component for use in iAPX 286 microsystems. The bus controller provides command and control outputs with flexible timing options. Separate command outputs are used for memory and I/O devices. The data bus is controlled with separate data enable and direction control signals.

Two modes of operation are possible via a strapping option: Multibus compatible bus cycles, and high speed bus cycles.

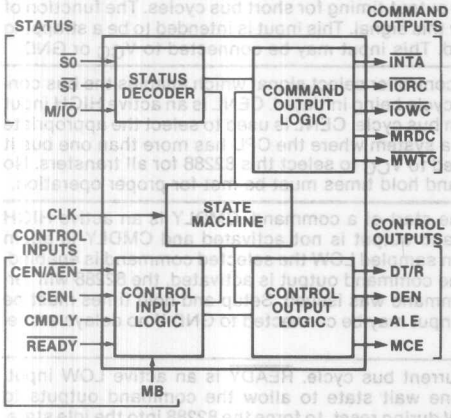


Figure 1. 82288 Block Diagram

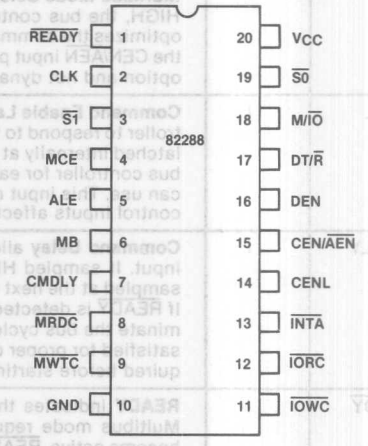


Figure 2. 82288 Pin Diagram

*Multibus is a patented bus of Intel.

Intel Corporation Assumes No Responsibility for the Use of Any Circuitry Other Than Circuitry Embodied in an Intel Product. No Other Circuit Patent Licenses are Implied.

©INTEL CORPORATION, 1982

JUNE 1982
Order Number: 210471-001

Table 1. Pin Description

The following pin function descriptions are for the 82288 bus controller.

Symbol	Type	Name and Function
CLK	I	**System Clock** provides the basic timing control for the 82288 in an iAPX 286 microsystem. Its frequency is twice the internal processor clock frequency. The falling edge of this input signal establishes when inputs are sampled and command and control outputs change.
$\overline{S0}$, $\overline{S1}$	I	**Bus Cycle Status** starts a bus cycle and, along with M/$\overline{IO}$, defines the type of bus cycle. These inputs are active LOW. A bus cycle is started when either $\overline{S1}$ or $\overline{S0}$ is sampled LOW at the falling edge of CLK. These inputs have pullups sufficient to hold them HIGH when nothing drives them. Setup and hold times must be met for proper operation.
M/$\overline{IO}$	I	**Memory or I/O Select** determines whether the current bus cycle is in the memory space or I/O space. When LOW, the current bus cycle is in the I/O space. Setup and hold times must be met for proper operation.
MB	I	**Multibus Mode Select** determines timing of the command and control outputs. When HIGH, the bus controller operates in Multibus mode. When LOW, the bus controller optimizes the command and control output timing for short bus cycles. The function of the CEN/$\overline{AEN}$ input pin is selected by this signal. This input is intended to be a strapping option and not dynamically changed. This input may be connected to V_{CC} or GND.
CENL	I	**Command Enable Latched** is a bus controller select signal which enables the bus controller to respond to the current bus cycle being initiated. CENL is an active HIGH input latched internally at the start of each bus cycle. CENL is used to select the appropriate bus controller for each bus cycle in a system where the CPU has more than one bus it can use. This input may be connected to V_{CC} to select this 82288 for all transfers. No control inputs affect CENL. Setup and hold times must be met for proper operation.
CMDLY	I	**Command Delay** allows delaying the start of a command. CMDLY is an active HIGH input. If sampled HIGH, the command output is not activated and CMDLY is again sampled at the next CLK cycle. When sampled LOW the selected command is enabled. If $\overline{READY}$ is detected LOW before the command output is activated, the 82288 will terminate the bus cycle, even if no command was issued. Setup and hold times must be satisfied for proper operation. This input may be connected to GND if no delays are required before starting a command.
$\overline{READY}$	I	**READY** indicates the end of the current bus cycle. $\overline{READY}$ is an active LOW input. Multibus mode requires at least one wait state to allow the command outputs to become active. $\overline{READY}$ must be LOW during reset, to force the 82288 into the idle state. Setup and hold times must be met for proper operation.

iAPX 286 Bus Cycle Status Definition

M/$\overline{IO}$	$\overline{S1}$	$\overline{S0}$	Type of Bus Cycle
0	0	0	Interrupt acknowledge
0	0	1	I/O Read
0	1	0	I/O Write
0	1	1	None; idle
1	0	0	Halt or shutdown
1	0	1	Memory read
1	1	0	Memory write
1	1	1	None; idle

AFN-00787A

Table 1. Pin Description (Cont.)

Symbol	Type	Name and Function
CEN/$\overline{AEN}$	I	**Command Enable/Address Enable** controls the command and DEN outputs of the bus controller. CEN/$\overline{AEN}$ inputs may be asynchronous to CLK. Setup and hold times are given to assure a guaranteed response to synchronous inputs. This input may be connected to V_{CC} or GND. When MB is HIGH this pin has the $\overline{AEN}$ function. $\overline{AEN}$ is an active LOW input which indicates that the CPU has been granted use of a shared bus and the bus controller command outputs may exit 3-state OFF and become inactive (HIGH). $\overline{AEN}$ HIGH indicates that the CPU does not have control of the shared bus and forces the command outputs into 3-state OFF and DEN inactive (LOW). $\overline{AEN}$ would normally be controlled by an 82289 bus arbiter which activates $\overline{AEN}$ when that arbiter owns the bus to which the bus controller is attached. When MB is LOW this pin has the CEN function. CEN is an unlatched active HIGH input which allows the bus controller activate its command and DEN outputs. With MB LOW, CEN LOW forces the command and DEN outputs inactive but does not tristate them.
ALE	O	**Address Latch Enable** controls the address latches used to hold an address stable during a bus cycle. This control output is active HIGH. ALE will not be issued for the halt bus cycle and is not affected by any of the control inputs.
MCE	O	**Master Cascade Enable** signals that a cascade address from a master 8259A interrupt controller may be placed onto the CPU address bus for latching by the address latches under ALE control. The CPU's address bus may then be used to broadcast the cascade address to slave interrupt controllers so only one of them will respond to the interrupt acknowledge cycle. This control output is active HIGH. MCE is only active during interrupt acknowledge cycles and is not affected by any control input. Using MCE to enable cascade address drivers requires latches which save the cascade address on the falling edge of ALE.
DEN	O	**Data Enable** controls when data transceivers connected to the local data bus should be enabled. DEN is an active HIGH control output. DEN is delayed for write cycles in the Multibus mode.
DT/$\overline{R}$	O	**Data Transmit/Receive** establishes the direction of data flow to or from the local data bus. When HIGH, this control output indicates that a write bus cycle is being performed. A LOW indicates a read bus cycle. DEN is always inactive when DT/$\overline{R}$ changes states. This output is HIGH when no bus cycle is active. DT/$\overline{R}$ is not affected by any of the control inputs.
$\overline{IOWC}$	O	**I/O Write Command** instructs an I/O device to read the data on the data bus. This command output is active LOW. The MB and CMDLY inputs control when this output becomes active. $\overline{READY}$ controls when it becomes inactive.
$\overline{IORC}$	O	**I/O Read Command** instructs an I/O device to place data onto the data bus. This command output is active LOW. The MB and CMDLY inputs control when this output becomes active. $\overline{READY}$ controls when it becomes inactive.
$\overline{MWTC}$	O	**Memory Write Command** instructs a memory device to read the data on the data bus. This command output is active LOW. The MB and CMDLY inputs control when this output becomes active. $\overline{READY}$ controls when it becomes inactive.
$\overline{MRDC}$	O	**Memory Read Command** instructs the memory device to place data onto the data bus. This command output is active LOW. The MB and CMDLY inputs control when this output becomes active. $\overline{READY}$ controls when it becomes inactive.
$\overline{INTA}$	O	**Interrupt Acknowledge** tells an interrupting device that its interrupt request is being acknowledged. This command output is active LOW. The MB and CMDLY inputs control when this output becomes active. $\overline{READY}$ controls when it becomes inactive.
V_{CC}		**System Power:** +5V power supply
GND		**System Ground:** 0 volts

AFN-00787A

FUNCTIONAL DESCRIPTION

Introduction

The 82288 bus controller is used in iAPX 286 systems to provide address latch control, data transceiver control, and standard level-type command outputs. The command outputs are timed and have sufficient drive capabilities for large TTL buses and meet all IEEE-796 requirements for Multibus. A special Multibus mode is provided to statisfy all address/data setup and hold time requirements. Command timing may be tailored to special needs via a CMDLY input to determine the start of a command and $\overline{\text{READY}}$ to determine the end of a command.

Connection to multiple buses are supported with a latched enable input (CENL). An address decoder can determine which, if any, bus controller should be enabled for the bus cycle. This input is latched to allow an address decoder to take full advantage of the pipelined timing on the iAPX 286 local bus.

Buses shared by several bus controllers are supported. An $\overline{\text{AEN}}$ input prevents the bus controller from driving the shared bus command and data signals except when enabled by an external bus arbiter such as the 82289.

Separate DEN and DT/$\overline{\text{R}}$ outputs control the data transceivers for all buses. Bus contention is eliminated by disabling DEN before changing DT/$\overline{\text{R}}$. The DEN timing allows sufficient time for tristate bus drivers to enter 3-state OFF before enabling other drivers onto the same bus.

The term CPU refers to any iAPX 286 processor or iAPX 286 support component which may become an iAPX 286 local bus master and thereby drive the 82288 status inputs.

Processor Cycle Definition

Any CPU which drives the local bus uses an internal clock which is one half the frequency of the system clock (CLK) (see Figure 3). Knowledge of the phase of the local bus master internal clock is required for proper operation of the iAPX 286 local bus. The local bus master informs the bus controller of its internal clock phase when it asserts the status signals. Status signals are always asserted in Phase 1 of the local bus master's internal clock.

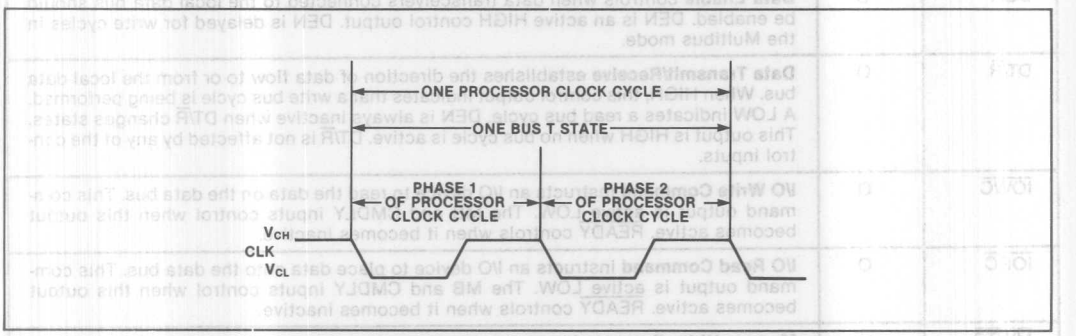


Figure 3. CLK Relationship to the Processor Clock and Bus T-States

AFN-00787A

Bus State Definition

The 82288 bus controller has three bus states (see Figure 4): Idle (T_I) Status (T_S) and Command (T_C). Each bus state is two CLK cycles long. Bus state phases correspond to the internal CPU processor clock phases.

The T_I bus state occurs when no bus cycle is currently active on the iAPX 286 local bus. This state may be repeated indefinitely. When control of the local bus is being passed between masters, the bus remains in the T_I state.

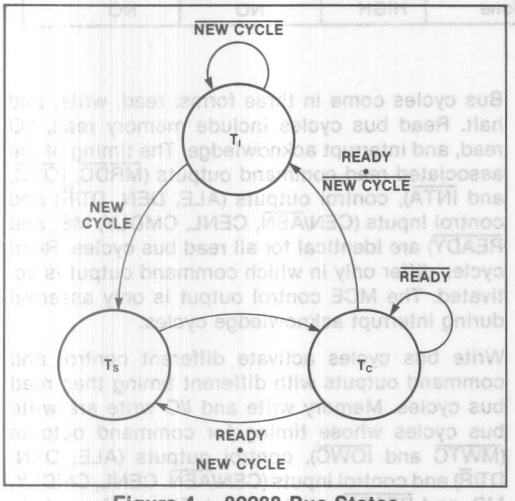


Figure 4. 82288 Bus States

Bus Cycle Definition

The $\overline{S1}$ and $\overline{S0}$ inputs signal the start of a bus cycle. When either input becomes LOW, a bus cycle is started. The T_S bus state is defined to be the two CLK cycles during which either $\overline{S1}$ or $\overline{S0}$ are active (see Figure 5). These inputs are sampled by the 82288 at every falling edge of CLK. When either $\overline{S1}$ or $\overline{S0}$ are sampled LOW, the next CLK cycle is considered the second phase of the internal CPU clock cycle.

The local bus enters the T_C bus state after the T_S state. The shortest bus cycle may have one T_S state and one T_C state. Longer bus cycles are formed by repeating T_C states. A repeated T_C bus state is called a wait state.

The $\overline{READY}$ input determines whether the current T_C bus state is to be repeated. The $\overline{READY}$ input has the same timing and effect for all bus cycles. $\overline{READY}$ is sampled at the end of each T_C bus state to see if it is active. If sampled HIGH, the T_C bus state is repeated. This is called inserting a wait state. The control and command outputs do not change during wait states.

When $\overline{READY}$ is sampled LOW, the current bus cycle is terminated. Note that the bus controller may enter the T_S bus state directly from T_C if the status lines are sampled active at the next falling edge of CLK.

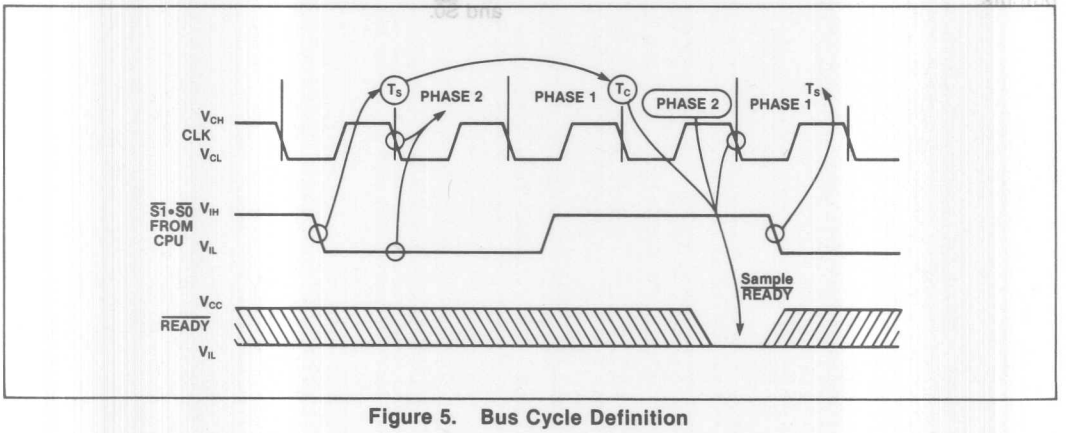


Figure 5. Bus Cycle Definition

AFN-00787A

Table 2. Command and Control Outputs for Each Type of Bus Cycle

Type of Bus Cycle	M/$\overline{IO}$	$\overline{S1}$	$\overline{S0}$	Command Activated	DT/$\overline{R}$ State	ALE, DEN Issued?	MCE Issued?
Interrupt Acknowledge	0	0	0	$\overline{INTA}$	LOW	YES	YES
I/O Read	0	0	1	$\overline{IORC}$	LOW	YES	NO
I/O Write	0	1	0	$\overline{IOWC}$	HIGH	YES	NO
None; idle	0	1	1	None	HIGH	NO	NO
Halt/Shutdown	1	0	0	None	HIGH	NO	NO
Memory Read	1	0	1	$\overline{MRDC}$	LOW	YES	NO
Memory Write	1	1	0	$\overline{MWTC}$	HIGH	YES	NO
None; idle	1	1	1	None	HIGH	NO	NO

Operating Modes

Two types of buses are supported by the 82288: Multibus and non-Multibus. When the MB input is strapped HIGH, Multibus timing is used. In Multibus mode, the 82288 delays command and data activation to meet IEEE-796 requirements on address to command active and write data to command active setup timing. Multibus mode requires at least one wait state in the bus cycle since the command outputs are delayed. The non-Multibus mode does not delay any outputs and does not require wait states. The MB input affects the timing of the command and DEN outputs.

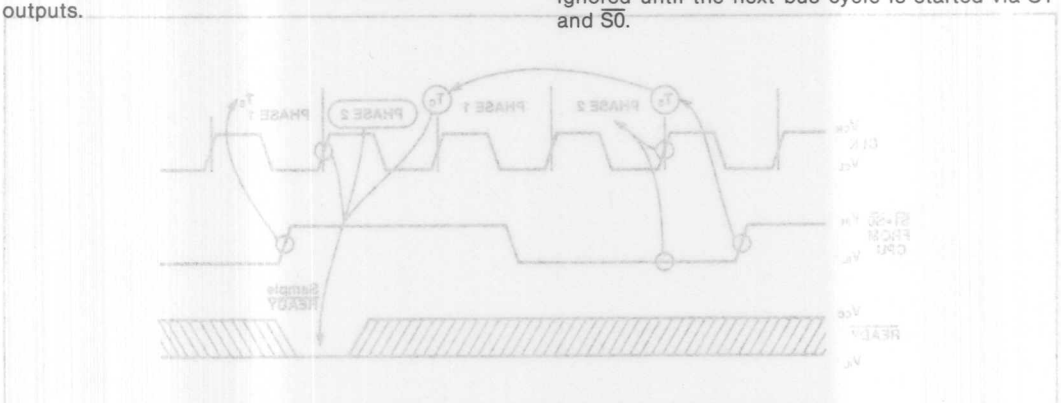
Command and Control Outputs

The type of bus cycle performed by the local bus master is encoded in the M/$\overline{IO}$, $\overline{S1}$, and $\overline{S0}$ inputs. Different command and control outputs are activated depending on the type of bus cycle. Table 2 indicates the cycle decode done by the 82288 and the effect on command, DT/$\overline{R}$, ALE, DEN, and MCE outputs.

Bus cycles come in three forms: read, write, and halt. Read bus cycles include memory read, I/O read, and interrupt acknowledge. The timing of the associated read command outputs ($\overline{MRDC}$, $\overline{IORC}$, and $\overline{INTA}$), control outputs (ALE, DEN, DT/$\overline{R}$) and control inputs (CEN/$\overline{AEN}$, CENL, CMDLY, MB, and $\overline{READY}$) are identical for all read bus cycles. Read cycles differ only in which command output is activated. The MCE control output is only asserted during interrupt acknowledge cycles.

Write bus cycles activate different control and command outputs with different timing than read bus cycles. Memory write and I/O write are write bus cycles whose timing for command outputs ($\overline{MWTC}$ and $\overline{IOWC}$), control outputs (ALE, DEN, DT/$\overline{R}$) and control inputs (CEN/$\overline{AEN}$, CENL, CMDLY, MB, and $\overline{READY}$) are identical. They differ only in which command output is activated.

Halt bus cycles are different because no command or control output is activated. All control inputs are ignored until the next bus cycle is started via $\overline{S1}$ and $\overline{S0}$.



AFN-00787A

Figures 6-10 show the basic command and control output timing for read and write bus cycles. Halt bus cycles are not shown since they activate no outputs. The basic idle-read-idle and idle-write-idle bus cycles are shown. The signal label CMD represents the appropriate command output for the bus cycle. For Figures 6-10, the CMDLY input is connected to GND and CENL to V_{CC}. The effects of CENL and CMDLY are described later in the section on control inputs.

Figures 6 and 7 show non-Multibus cycles. MB is connected to GND while CEN is connected to V_{CC}. Figure 6 shows a read cycle with no wait states while Figure 7 shows a write cycle with one wait state. The $\overline{READY}$ input is shown to illustrate how wait states are added.

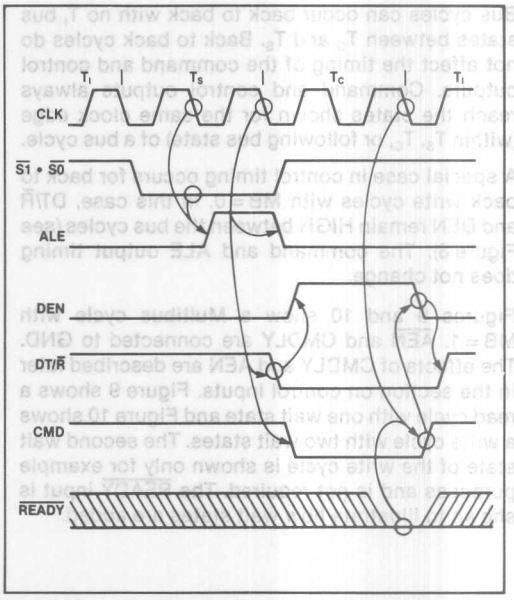


Figure 6. Idle-Read-Idle Bus Cycles with MB = 0

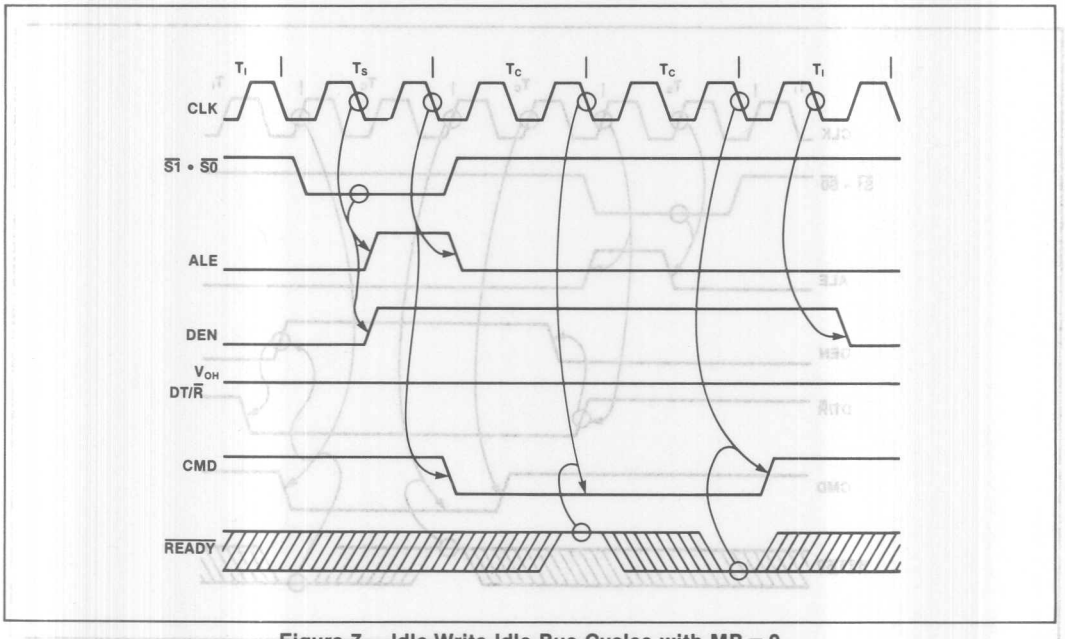


Figure 7. Idle-Write-Idle Bus Cycles with MB = 0

AFN-00787A

Bus cycles can occur back to back with no T_I bus states between T_C and T_S. Back to back cycles do not affect the timing of the command and control outputs. Command and control outputs always reach the states shown for the same clock edge (within T_S, T_C, or following bus state) of a bus cycle.

A special case in control timing occurs for back to back write cycles with MB = 0. In this case, DT/$\overline{R}$ and DEN remain HIGH between the bus cycles (see Figure 8). The command and ALE output timing does not change.

Figures 9 and 10 show a Multibus cycle with MB = 1. $\overline{AEN}$ and CMDLY are connected to GND. The effects of CMDLY and $\overline{AEN}$ are described later in the section on control inputs. Figure 9 shows a read cycle with one wait state and Figure 10 shows a write cycle with two wait states. The second wait state of the write cycle is shown only for example purposes and is not required. The $\overline{READY}$ input is shown to illustrate how wait states are added.

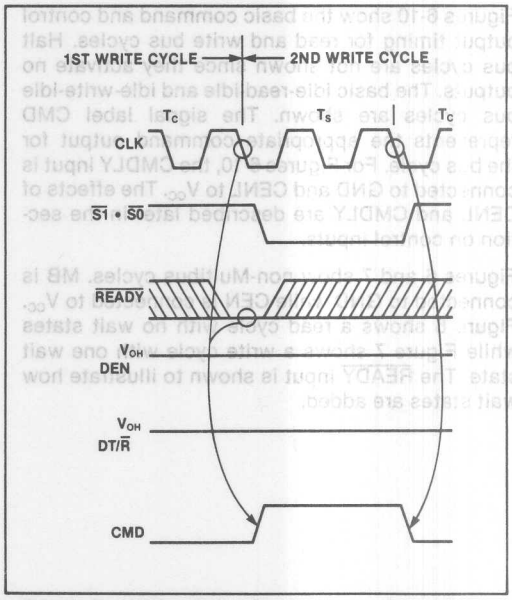


Figure 8. Write-Write Bus Cycles with MB = 0

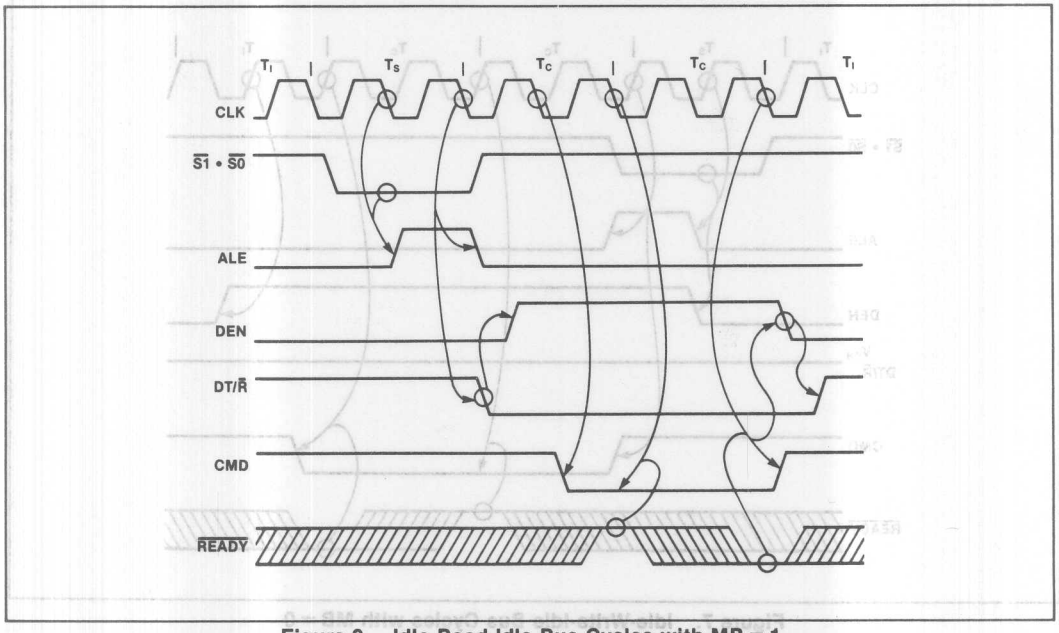


Figure 9. Idle-Read-Idle Bus Cycles with MB = 1

AFN-00787A

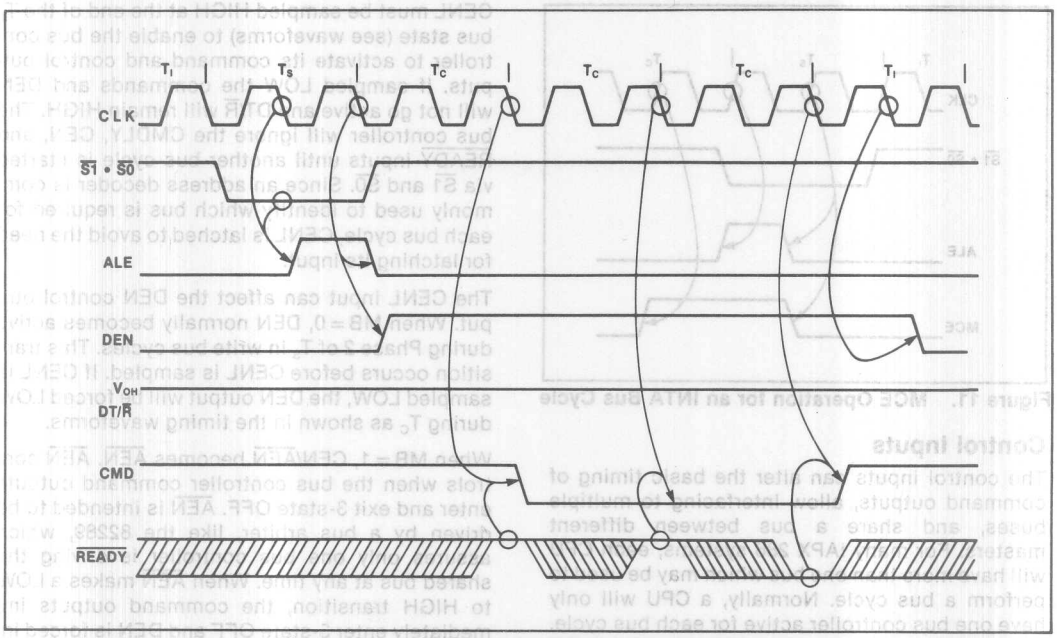


Figure 10. Idle-Write-Idle Bus Cycles with MB = 1

The MB control input affects the timing of the command and DEN outputs. These outputs are automatically delayed in Multibus mode to satisfy three requirements:

1) 50 ns minimum setup time for valid address before any command output becomes active.
2) 50 ns minimum setup time for valid write data before any write command output becomes active.
3) 65 ns maximum time from when any read command becomes inactive until the slave's read data drivers reach 3-state OFF.

Three signal transitions are delayed by MB = 1 as compared to MB = 0:

1) The HIGH to LOW transition of the read command outputs ($\overline{IORC}$, $\overline{MRDC}$, and $\overline{INTA}$) are delayed one CLK cycle.
2) The HIGH to LOW transition of the write command outputs ($\overline{IOWC}$ and $\overline{MWTC}$) are delayed two CLK cycles.
3) The LOW to HIGH transition of DEN for write cycles is delayed one CLK cycle.

Back to back bus cycles with MB = 1 do not change the timing of any of the command or control outputs. DEN always becomes inactive between bus cycles with MB = 1.

Except for a halt or shutdown bus cycle, ALE will be issued during the second half of T_S for any bus cycle. ALE becomes inactive at the end of the T_S to allow latching the address to keep it stable during the entire bus cycle. The address outputs may change during Phase 2 of any T_C bus state. ALE is not affected by any control input.

Figure 11 shows how MCE is timed during interrupt acknowledge (INTA) bus cycles. MCE is one CLK cycle longer than ALE to hold the cascade address from a master 8259A valid after the falling edge of ALE. With the exception of the MCE control output, an INTA bus cycle is identical in timing to a read bus cycle. MCE is not affected by any control input.

AFN-00787A

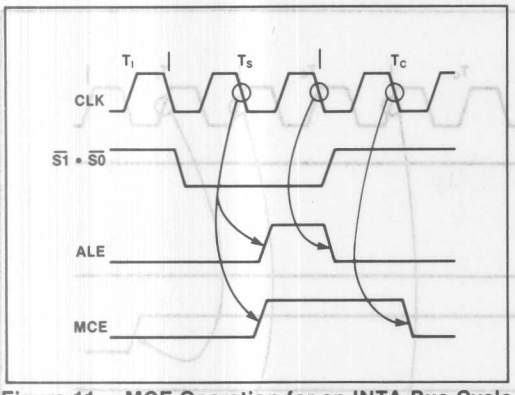


Figure 11. MCE Operation for an INTA Bus Cycle

Control Inputs

The control inputs can alter the basic timing of command outputs, allow interfacing to multiple buses, and share a bus between different masters. For many iAPX 286 systems, each CPU will have more than one bus which may be used to perform a bus cycle. Normally, a CPU will only have one bus controller active for each bus cycle. Some buses may be shared by more than one CPU (i.e. Multibus) requiring only one of them use the bus at a time.

Systems with multiple and shared buses use two control input signals of the 82288 bus controller, CENL and $\overline{AEN}$ (see Figure 12). CENL enables the bus controller to control the current bus cycle. The $\overline{AEN}$ input prevents a bus controller from driving its command outputs. $\overline{AEN}$ HIGH means that another bus controller may be driving the shared bus.

In Figure 12, two buses are shown: a local bus and a Multibus. Only one bus is used for each CPU bus cycle. The CENL inputs of the bus controllers select which bus controller is to perform the bus cycle. An address decoder determines which bus to use for each bus cycle. The 82288 connected to the shared Multibus must be selected by CENL and be given access to the Multibus by $\overline{AEN}$ before it will begin a Multibus operation.

CENL must be sampled HIGH at the end of the T_S bus state (see waveforms) to enable the bus controller to activate its command and control outputs. If sampled LOW the commands and DEN will not go active and DT/$\overline{R}$ will remain HIGH. The bus controller will ignore the CMDLY, CEN, and $\overline{READY}$ inputs until another bus cycle is started via $\overline{S1}$ and $\overline{S0}$. Since an address decoder is commonly used to identify which bus is required for each bus cycle, CENL is latched to avoid the need for latching its input.

The CENL input can affect the DEN control output. When MB = 0, DEN normally becomes active during Phase 2 of T_S in write bus cycles. This transition occurs before CENL is sampled. If CENL is sampled LOW, the DEN output will be forced LOW during T_C as shown in the timing waveforms.

When MB = 1, CEN/$\overline{AEN}$ becomes $\overline{AEN}$. $\overline{AEN}$ controls when the bus controller command outputs enter and exit 3-state OFF. $\overline{AEN}$ is intended to be driven by a bus arbiter, like the 82289, which assures only one bus controller is driving the shared bus at any time. When $\overline{AEN}$ makes a LOW to HIGH transition, the command outputs immediately enter 3-state OFF and DEN is forced inactive. An inactive DEN should force the local data transceivers connected to the shared data bus into 3-state OFF (see Figure 12). The LOW to HIGH transition of $\overline{AEN}$ should only occur during T_I or T_S bus states.

The HIGH to LOW transition of $\overline{AEN}$ signals that the bus controller may now drive the shared bus command signals. Since a bus cycle may be active or be in the process of starting, $\overline{AEN}$ can become active during any T-state. $\overline{AEN}$ LOW immediately allows DEN to go to the appropriate state. Three CLK edges later, the command outputs will go active (see timing waveforms). The Multibus requires this delay for the address and data to be valid on the bus before the commands become active.

When MB = 0, CEN/$\overline{AEN}$ becomes CEN. CEN is an asynchronous input which immediately affects the command and DEN outputs. When CEN makes a HIGH to LOW transition, the commands

AFN-00787A

and DEN are immediately forced inactive. When CEN makes a LOW to HIGH transition, the commands and DEN outputs immediately go to the appropriate state (see timing waveforms). READY must still become active to terminate a bus cycle if CEN remains LOW for a selected bus controller (CENL was latched HIGH).

Some memory or I/O systems may require more address or write data setup time to command active than provided by the basic command output timing. To provide flexible command timing, the CMDLY input can delay the activation of command outputs. The CMDLY input must be sampled LOW to activate the command outputs. CMDLY does not affect the control outputs ALE, MCE, DEN, and DT/R.

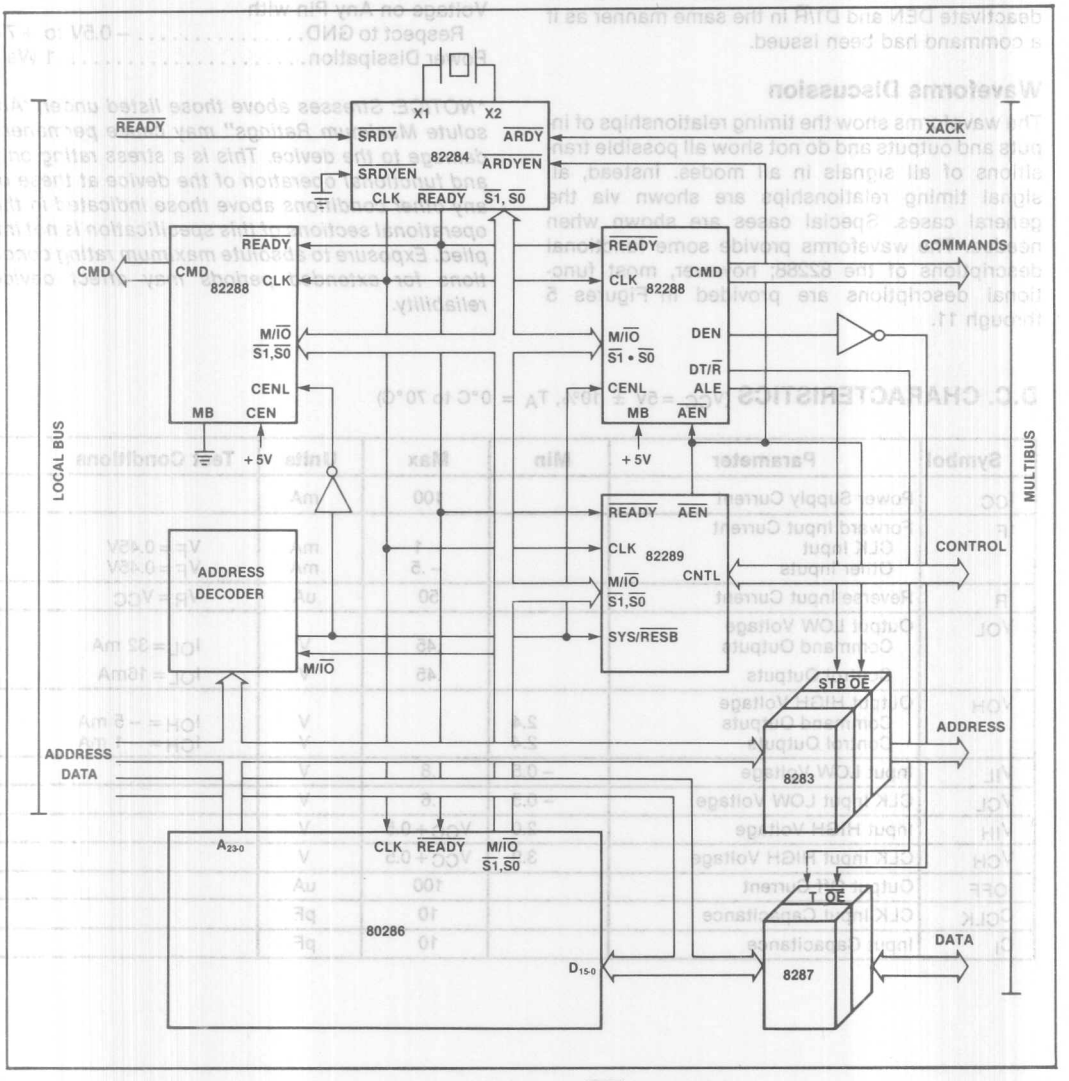


Figure 12. System Use of $\overline{AEN}$ and CENL

AFN-00787A

CMDLY is first sampled on the falling edge of the CLK ending T_S. If sampled HIGH, the command output is not activated, and CMDLY is again sampled on the next falling edge of CLK. Once sampled LOW, the proper command output becomes active immediately if MB = 0. If MB = 1, the proper command goes active no earlier than shown in Figures 9 and 10.

$\overline{\text{READY}}$ can terminate a bus cycle before CMDLY allows a command to be issued. In this case no commands are issued and the bus controller will deactivate DEN and DT/$\overline{\text{R}}$ in the same manner as if a command had been issued.

Waveforms Discussion

The waveforms show the timing relationships of inputs and outputs and do not show all possible transitions of all signals in all modes. Instead, all signal timing relationships are shown via the general cases. Special cases are shown when needed. The waveforms provide some functional descriptions of the 82288; however, most functional descriptions are provided in Figures 5 through 11.

To find the timing specification for a signal transition in a particular mode, first look for a special case in the waveforms. If no special case applies, then use a timing specification for the same or related function in another mode.

ABSOLUTE MAXIMUM RATINGS*

Ambient Temperature Under Bias. . . . 0°C to 70°C
Storage Temperature. −65°C to +150°C
Voltage on Any Pin with
Respect to GND. −0.5V to +7V
Power Dissipation. 1 Watt

**NOTICE: Stresses above those listed under "Absolute Maximum Ratings" may cause permanent damage to the device. This is a stress rating only and functional operation of the device at these or any other conditions above those indicated in the operational sections of this specification is not implied. Exposure to absolute maximum rating conditions for extended periods may affect device reliability.*

D.C. CHARACTERISTICS (V_{CC} = 5V ± 10%, T_A = 0°C to 70°C)

Symbol	Parameter	Min	Max	Units	Test Conditions
I_{CC}	Power Supply Current		100	mA	
I_F	Forward Input Current				
	CLK Input		−1	mA	V_F = 0.45V
	Other Inputs		−.5	mA	V_F = 0.45V
I_R	Reverse Input Current		50	uA	$V_R = V_{CC}$
V_{OL}	Output LOW Voltage				
	Command Outputs		.45	V	I_{OL} = 32 mA
	Control Outputs		.45	V	I_{OL} = 16mA
V_{OH}	Output HIGH Voltage				
	Command Outputs	2.4		V	I_{OH} = −5 mA
	Control Outputs	2.4		V	I_{OH} = −1 mA
V_{IL}	Input LOW Voltage	−0.5	.8	V	
V_{CL}	CLK Input LOW Voltage	−0.5	.6	V	
V_{IH}	Input HIGH Voltage	2.0	V_{CC} + 0.5	V	
V_{CH}	CLK Input HIGH Voltage	3.9	V_{CC} + 0.5	V	
I_{OFF}	Output Off Current		100	uA	
C_{CLK}	CLK Input Capacitance		10	pF	
C_I	Input Capacitance		10	pF	

AFN-00787A

A.C. CHARACTERISTICS (V_{CC} = 5V ± 10%, T_A = 0°C to 70°C)

Symbol	Parameter	Min	Max	Units	Test Conditions
t_1	CLK Period	62.5	250	ns	at 1.5V
t_2	CLK HIGH Time	20	235	ns	at 3.9V
t_3	CLK LOW Time	15	230	ns	at 0.6V
t_4	CLK Fall Time		10	ns	3.5V to 1.0V
t_5	CLK Rise Time		10	ns	1.0V to 3.5V
t_6	M/$\overline{IO}$ and Status Setup Time	22.5		ns	From 0.8V or 2.0V on input to 0.8V on CLK
t_7	M/$\overline{IO}$ and Status Hold Time	0		ns	
t_8	CENL Setup Time	20		ns	
t_9	CENL Hold Time	0		ns	
t_{10}	$\overline{READY}$ Setup Time	38.5		ns	
t_{11}	$\overline{READY}$ Hold Time	25		ns	
t_{12}	CMDLY Setup Time	20		ns	
t_{13}	CMDLY Hold Time	0		ns	
t_{14}	$\overline{AEN}$ Setup Time	25		ns	Note 1
t_{15}	$\overline{AEN}$ Hold Time	0		ns	Note 1
t_{16}	ALE, MCE Active Delay	3	15	ns	From 0.8V on CLK to 0.8V or 2.0V on output I_{OL} = 16 ma I_{OH} = – 1 ma C_L = 150 pF
t_{17}	ALE, MCE Inactive Delay		20	ns	
t_{18}	DEN (WRITE) Inactive From CENL		35	ns	
t_{19}	DT/$\overline{R}$ LOW From CLK		20	ns	
t_{20}	DEN (READ) Active From DT/$\overline{R}$	10	50	ns	
t_{21}	DEN (READ) Inactive Delay	3	15	ns	
t_{22}	DT/$\overline{R}$ HIGH From DEN Inactive	10	40	ns	
t_{23}	DEN (WRITE) Active Delay		30	ns	
t_{24}	DEN (WRITE) Inactive Delay	3	30	ns	
t_{25}	DEN Inactive From CEN		25	ns	
t_{26}	DEN Active From CEN		25	ns	
t_{27}	DT/$\overline{R}$ HIGH From CLK And CEN		50	ns	Note 2
t_{28}	DEN Active From $\overline{AEN}$		30	ns	I_{OL} = 32 ma I_{OH} = – 5 ma C_L = 300 pF From 0.8V on CLK to 0.8V or 2.0V on output
t_{29}	Command Active Delay	3	20	ns	
t_{30}	Command Inactive Delay	3	15	ns	
t_{31}	Command Inactive From CEN		25	ns	
t_{32}	Command Active From CEN		25	ns	
t_{33}	Command Inactive Enable From $\overline{AEN}$		40	ns	
t_{34}	Command Float Time		40	ns	

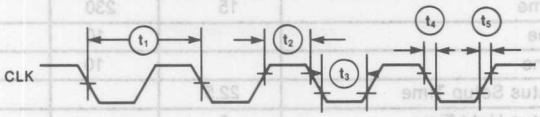
Note 1: $\overline{AEN}$ is a asynchronous input. $\overline{AEN}$ setup and hold time is specified to guarantee the response shown in the waveforms.

Note 2: T_{27} only applies to bus cycles where MB = 0, the 82288 was selected, and DEN = 0 when the cycle terminated (because CEN = 0).

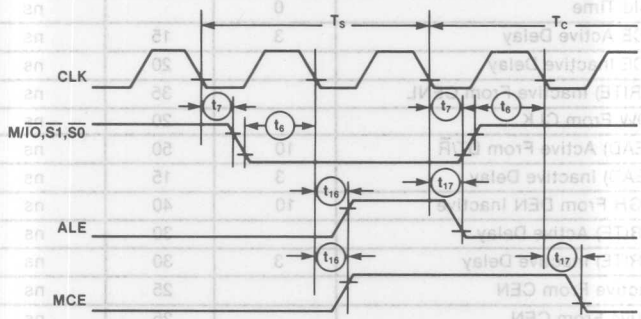
AFN-00787A

WAVEFORMS

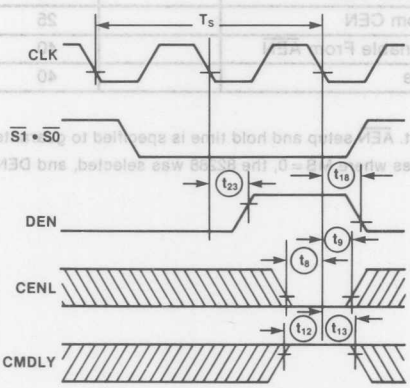
CLK CHARACTERISTICS



STATUS, ALE, MCE, CHARACTERISTICS



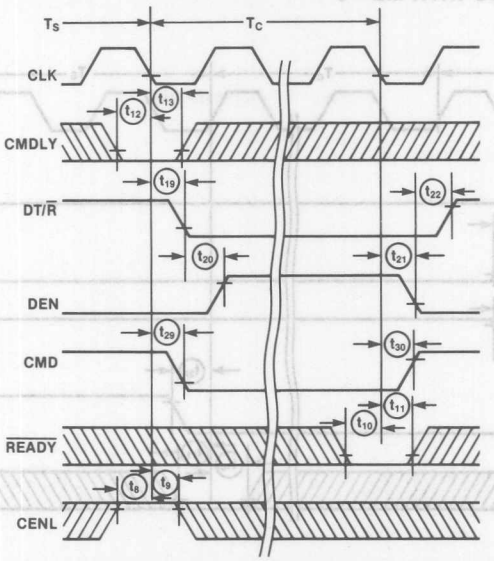
CENL, CMDLY, DEN CHARACTERISTICS WITH MB = 0 AND CEN = 1 DURING WRITE CYCLE



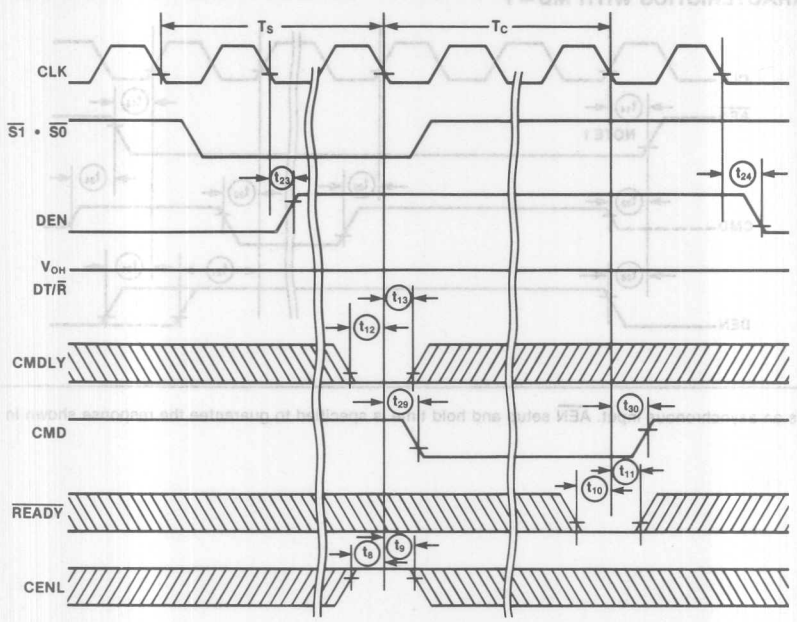
AFN-00787A

WAVEFORMS (Continued)

READ CYCLE CHARACTERISTICS WITH MB = 0 AND CEN = 1

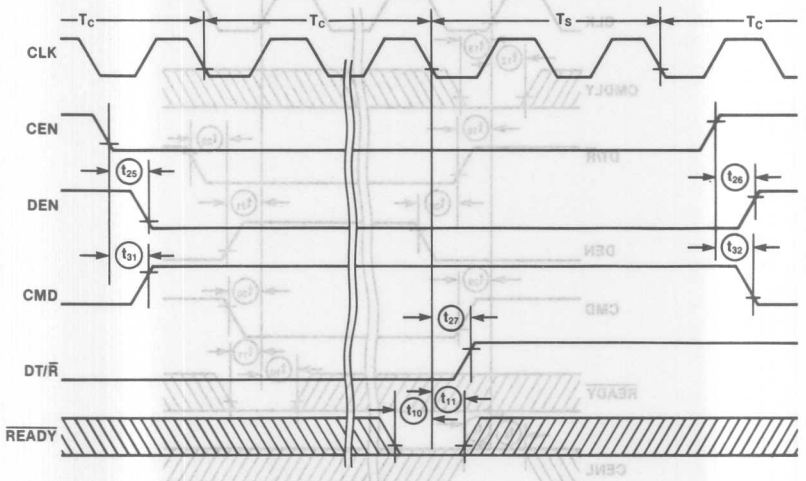


WRITE CYCLE CHARACTERISTICS WITH MB = 0 AND CEN = 1

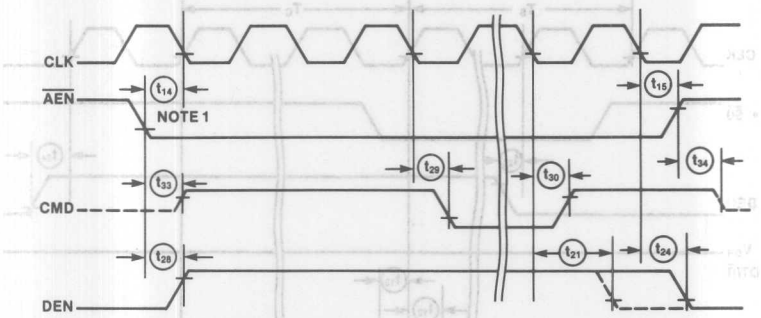


AFN-00787A

CEN CHARACTERISTICS WITH MB = 0



AEN CHARACTERISTICS WITH MB = 1



NOTE 1: AEN is an asynchronous input. AEN setup and hold time is specified to guarantee the response shown in the waveforms.

8282/8283
OCTAL LATCH

- **Address Latch for iAPX 86, 88, 186, 188, MCS-80®, MCS-85®, MCS-48® Famlies**
- **High Output Drive Capability for Driving System Data Bus**
- **Fully Parallel 8-Bit Data Register and Buffer**
- **Transparent during Active Strobe**
- **3-State Outputs**
- **20-Pin Package with 0.3" Center**
- **No Output Low Noise when Entering or Leaving High Impedance State**
- **Available in EXPRESS**
 - **Standard Temperature Range**
 - **Extended Temperature Range**

The 8282 and 8283 are 8-bit bipolar latches with 3-state output buffers. They can be used to implement latches, buffers, or multiplexers. The 8283 inverts the input data at its outputs while the 8282 does not. Thus, all of the principal peripheral and input/output functions of a microcomputer system can be implemented with these devices.

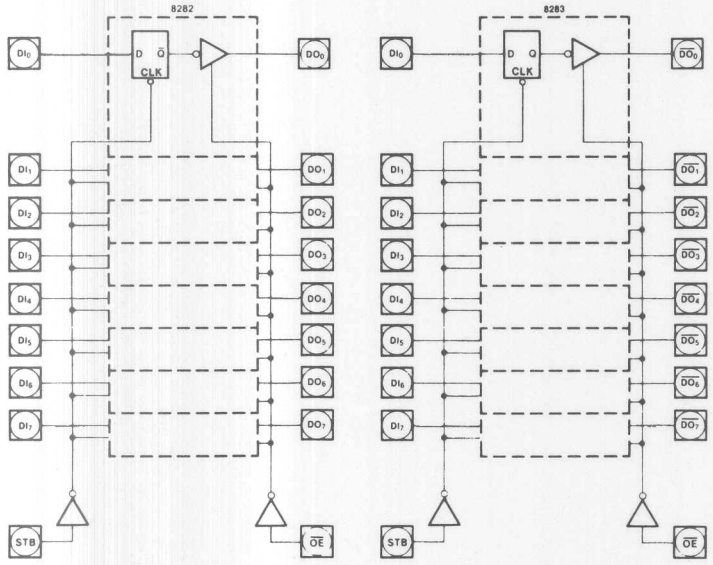


Figure 1. Logic Diagrams

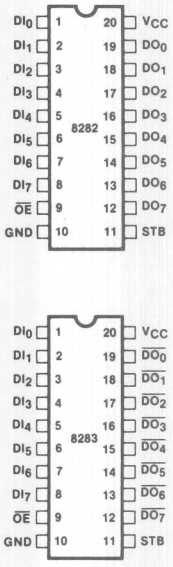


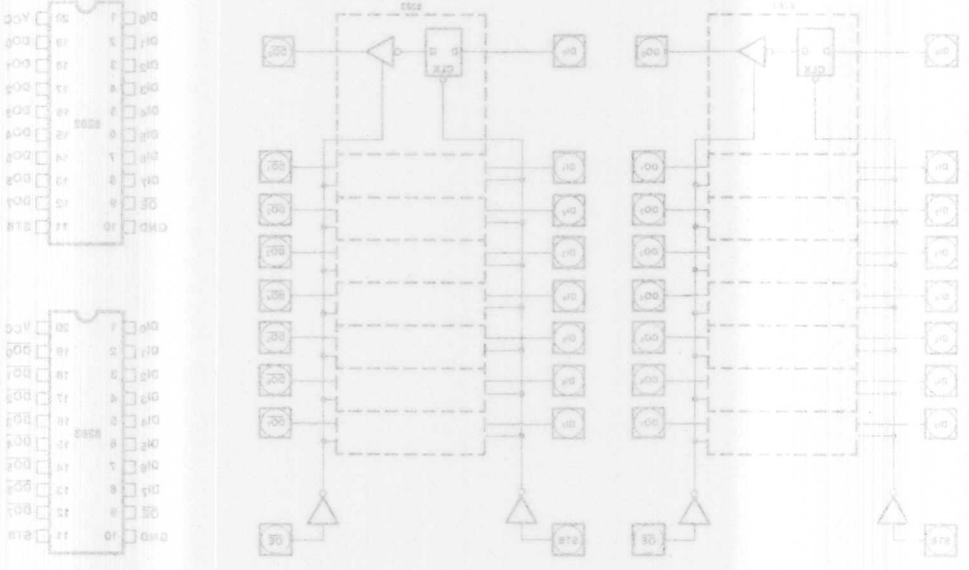
Figure 2. Pin Configurations

Table 1. Pin Description

Pin	Description
STB	STROBE (Input). STB is an input control pulse used to strobe data at the data input pins (A_0-A_7) into the data latches. This signal is active HIGH to admit input data. The data is latched at the HIGH to LOW transition of STB.
$\overline{OE}$	OUTPUT ENABLE (Input). $\overline{OE}$ is an input control signal which when active LOW enables the contents of the data latches onto the data output pin (B_0-B_7). OE being inactive HIGH forces the output buffers to their high impedance state.
DI_0-DI_7	DATA INPUT PINS (Input). Data presented at these pins satisfying setup time requirements when STB is strobed and latched into the data input latches.
DO_0-DO_7 (8282) $\overline{DO_0}$-$\overline{DO_7}$ (8283)	DATA OUTPUT PINS (Output). When $\overline{OE}$ is true, the data in the data latches is presented as inverted (8283) or non-inverted (8282) data onto the data output pins.

FUNCTIONAL DESCRIPTION

The 8282 and 8283 octal latches are 8-bit latches with 3-state output buffers. Data having satisfied the setup time requirements is latched into the data latches by strobing the STB line HIGH to LOW. Holding the STB line in its active HIGH state makes the latches appear transparent. Data is presented to the data output pins by activating the $\overline{OE}$ input line. When $\overline{OE}$ is inactive HIGH the output buffers are in their high impedance state. Enabling or disabling the output buffers will not cause negative-going transients to appear on the data output bus.



AFN-00727E

ABSOLUTE MAXIMUM RATINGS*

Temperature Under Bias 0°C to 70°C
Storage Temperature −65°C to +150°C
All Output and Supply Voltages −0.5V to +7V
All Input Voltages −1.0V to +5.5V
Power Dissipation 1 Watt

NOTICE: Stresses above those listed under "Absolute Maximum Ratings" may cause permanent damage to the device. This is a stress rating only and functional operation of the device at these or any other conditions above those indicated in the operational sections of this specification is not implied. Exposure to absolute maximum rating conditions for extended periods may affect device reliability.

D.C. CHARACTERISTICS (V_{CC} = 5V ±10%, T_A = 0°C to 70°C)

Symbol	Parameter	Min.	Max.	Units	Test Conditions
V_C	Input Clamp Voltage		−1	V	I_C = −5 mA
I_{CC}	Power Supply Current		160	mA	
I_F	Forward Input Current		−0.2	mA	V_F = 0.45V
I_R	Reverse Input Current		50	μA	V_R = 5.25V
V_{OL}	Output Low Voltage		.45	V	I_{OL} = 32 mA
V_{OH}	Output High Voltage	2.4		V	I_{OH} = −5 mA
I_{OFF}	Output Off Current		±50	μA	V_{OFF} = 0.45 to 5.25V
V_{IL}	Input Low Voltage		0.8	V	V_{CC} = 5.0V See Note 1
V_{IH}	Input High Voltage	2.0		V	V_{CC} = 5.0V See Note 1
C_{IN}	Input Capacitance		12	pF	F = 1 MHz V_{BIAS} = 2.5V, V_{CC} = 5V T_A = 25°C

NOTE:
1. Output Loading I_{OL} = 32 mA, I_{OH} = −5 mA, C_L = 300 pF.*

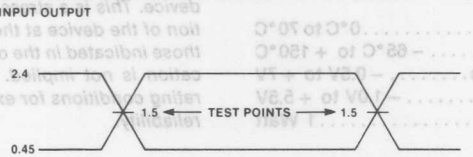
A.C. CHARACTERISTICS (V_{CC} = 5V ±10%, T_A = 0°C to 70°C (See Note 2)
Loading: Outputs—I_{OL} = 32 mA, I_{OH} = −5 mA, C_L = 300 pF*)

Symbol	Parameter	Min.	Max.	Units	Test Conditions
TIVOV	Input to Output Delay				(See Note 1)
	—Inverting	5	22	ns	
	—Non-Inverting	5	30	ns	
TSHOV	STB to Output Delay				
	—Inverting	10	40	ns	
	—Non-Inverting	10	45	ns	
TEHOZ	Output Disable Time	5	18	ns	
TELOV	Output Enable Time	10	30	ns	
TIVSL	Input to STB Setup Time	0		ns	
TSLIX	Input to STB Hold Time	25		ns	
TSHSL	STB High Time	15		ns	
TOLOH	Input, Output Rise Time		20	ns	From 0.8V to 2.0V
TOHOL	Input, Output Fall Time		12	ns	From 2.0V to 0.8V

NOTE:
*C_L = 200 pF for plastic 8282/8283.
1. See waveforms and test load circuit on following page.
2. For Extended Temperature EXPRESS the Preliminary Maximum Values are TIVOV = 25 vs 22, 35 vs 30; TSHOV = 45, 55; TEHOZ = 25; TELOV = 50.

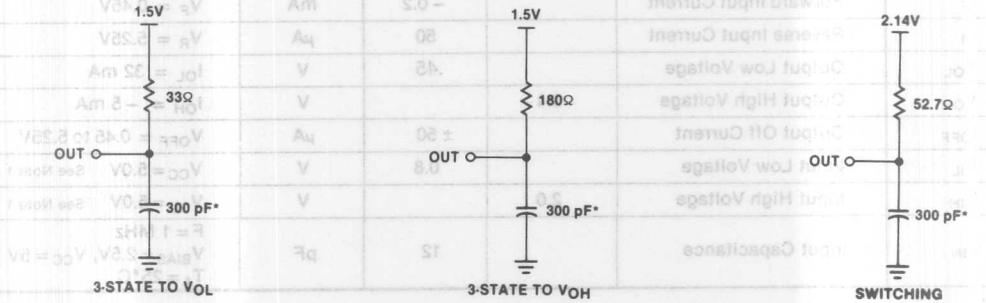
AFN-00727E

A.C. TESTING INPUT, OUTPUT WAVEFORM



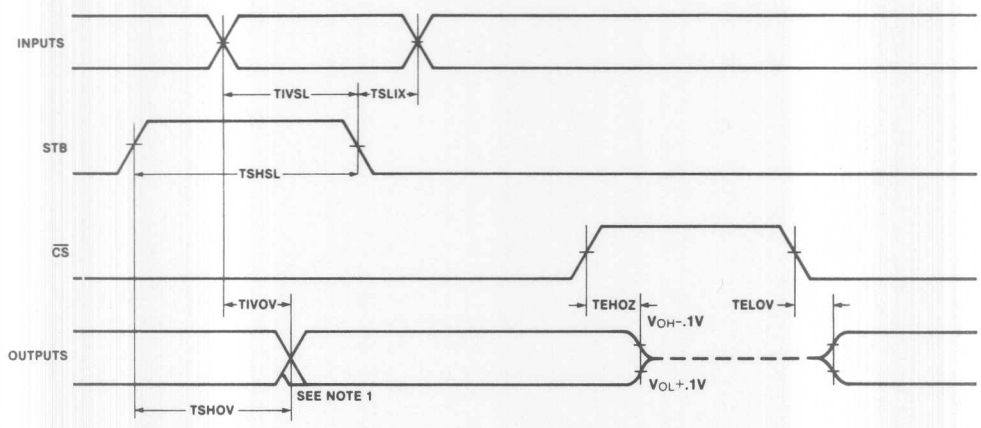
A.C. TESTING: INPUTS ARE DRIVEN AT 2.4V FOR A LOGIC "1" AND 0.45V FOR A LOGIC "0." TIMING MEASUREMENTS ARE MADE AT 1.5V FOR BOTH A LOGIC "1" AND "0." INPUT RISE AND FALL TIMES ARE MEASURED FROM 0.8V TO 2.0V AND ARE DRIVEN AT 5ns ± 2ns.

OUTPUT TEST LOAD CIRCUITS

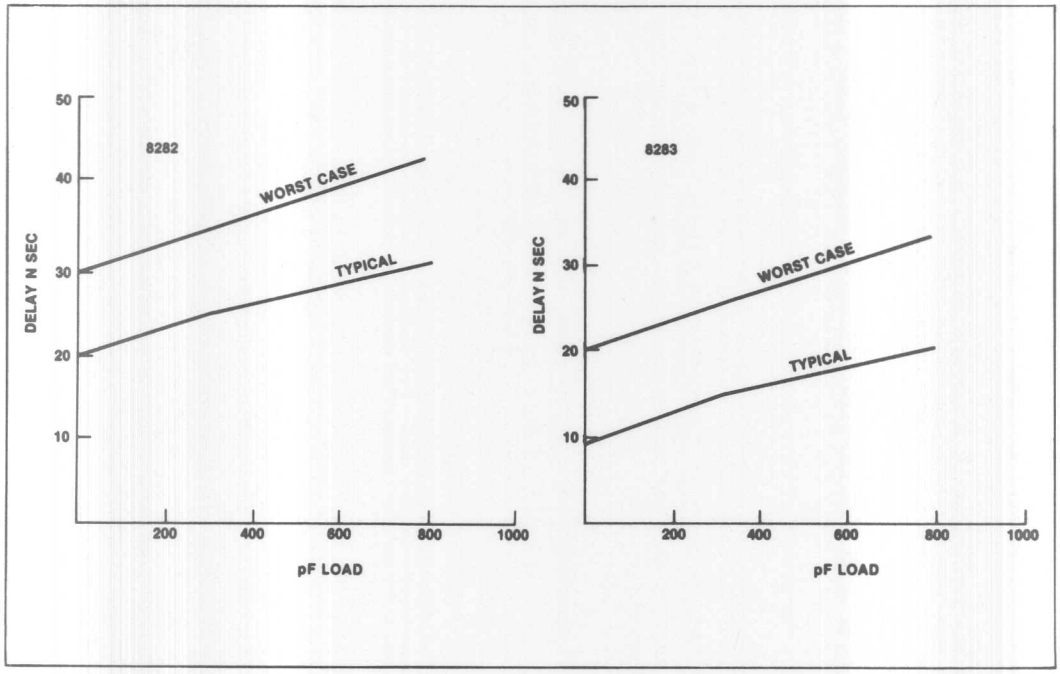


*200 pF for plastic 8282/8283.

WAVEFORMS

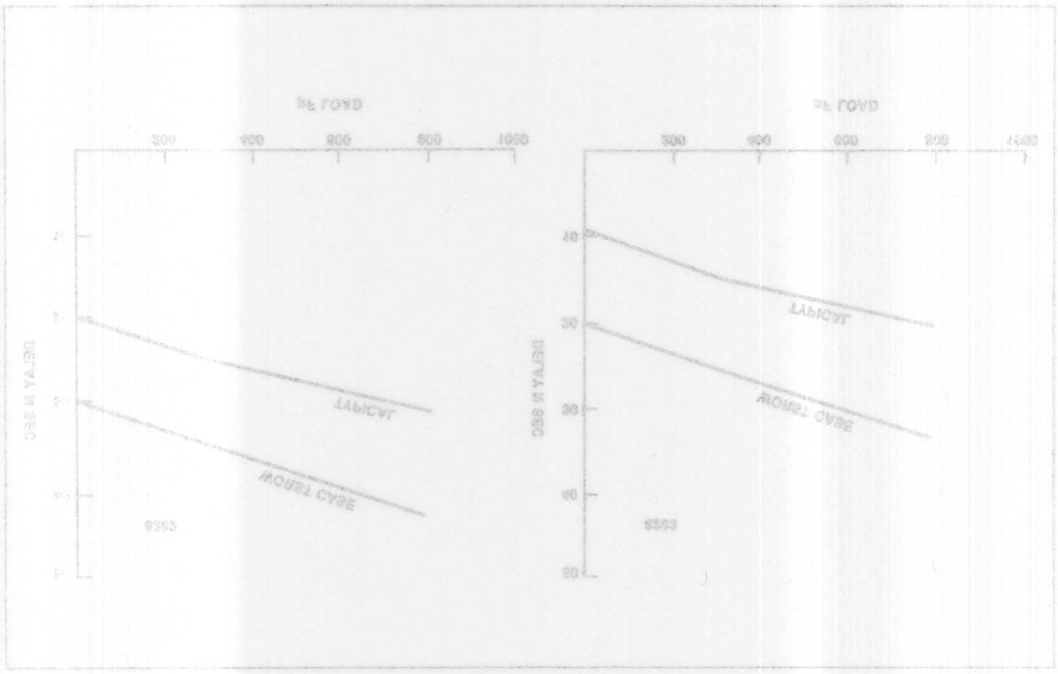


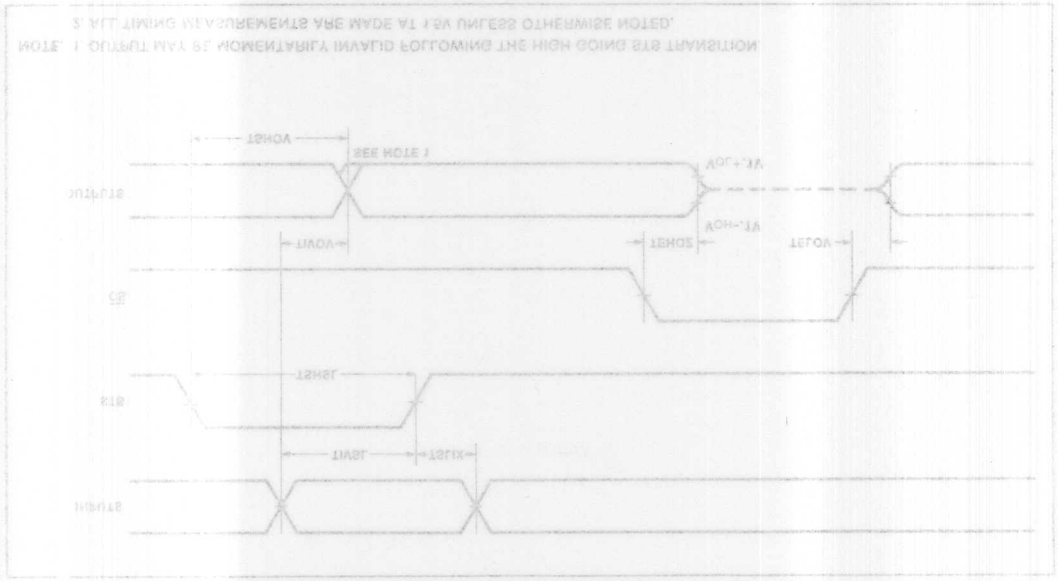
NOTE: 1. OUTPUT MAY BE MOMENTARILY INVALID FOLLOWING THE HIGH GOING STB TRANSITION.
2. ALL TIMING MEASUREMENTS ARE MADE AT 1.5V UNLESS OTHERWISE NOTED.



Output Delay vs. Capacitance

AFN-00727E


Output Delay vs. Capacitance
TYPICAL
WORST CASE
TYPICAL
WORST CASE


SEE NOTE 1
SB
RIB

WAVEFORMS

8286/8287
OCTAL BUS TRANSCEIVER

- **Data Bus Buffer Driver for iAPX 86,88,186,188, MCS-80™, MCS-85™, and MCS-48™ Families**
- **High Output Drive Capability for Driving System Data Bus**
- **Fully Parallel 8-Bit Transceivers**
- **3-State Outputs**
- **20-Pin Package with 0.3" Center**
- **No Output Low Noise when Entering or Leaving High Impedance State**
- **Available in EXPRESS**
 - **Standard Temperature Range**
 - **Extended Temperature Range**

The 8286 and 8287 are 8-bit bipolar transceivers with 3-state outputs. The 8287 inverts the input data at its outputs while the 8286 does not. Thus, a wide variety of applications for buffering in microcomputer systems can be met.

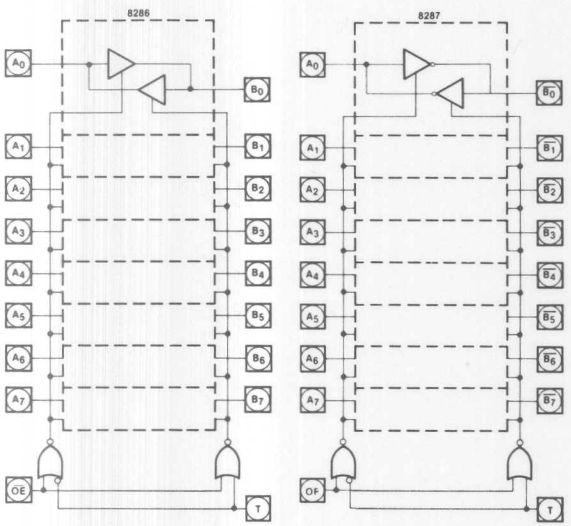


Figure 1. Logic Diagrams

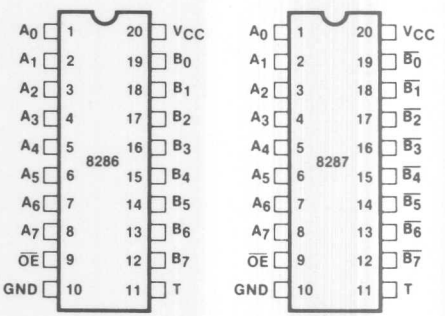


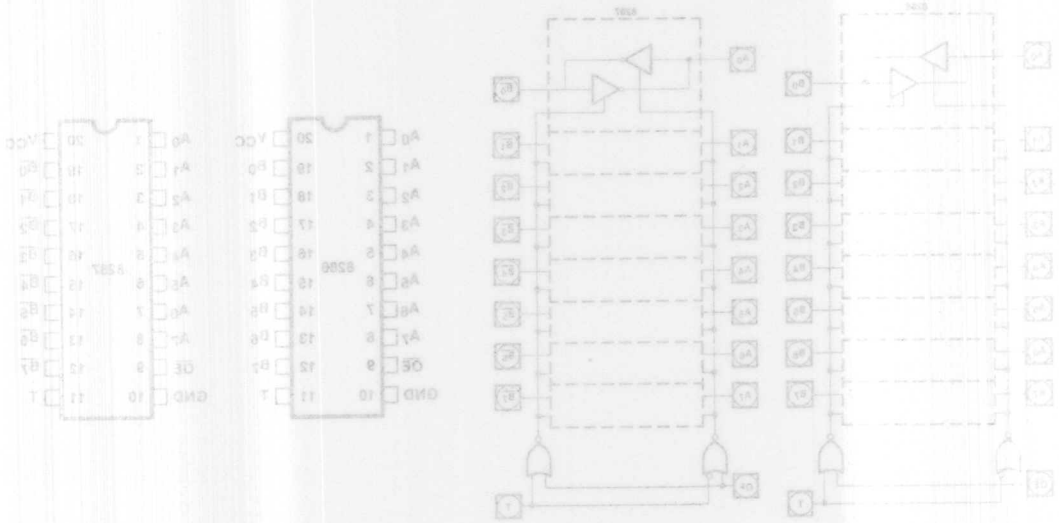
Figure 2. Pin Configurations

Table 1. Pin Description

Symbol	Type	Name and Function
T	I	**Transmit:** T is an input control signal used to control the direction of the transceivers. When HIGH, it configures the transceiver's B_0–B_7 as outputs with A_0–A_7 as inputs. T LOW configures A_0–A_7 as the outputs with B_0–B_7 serving as the inputs.
$\overline{OE}$	I	**Output Enable:** $\overline{OE}$ is an input control signal used to enable the appropriate output driver (as selected by T) onto its respective bus. This signal is active LOW.
A_0–A_7	I/O	**Local Bus Data Pins:** These pins serve to either present data to or accept data from the processor's local bus depending upon the state of the T pin.
B_0–B_7(8286) $\overline{B_0}$–$\overline{B_7}$(8287)	I/O	**System Bus Data Pins:** These pins serve to either present data to or accept data from the system bus depending upon the state of the T pin.

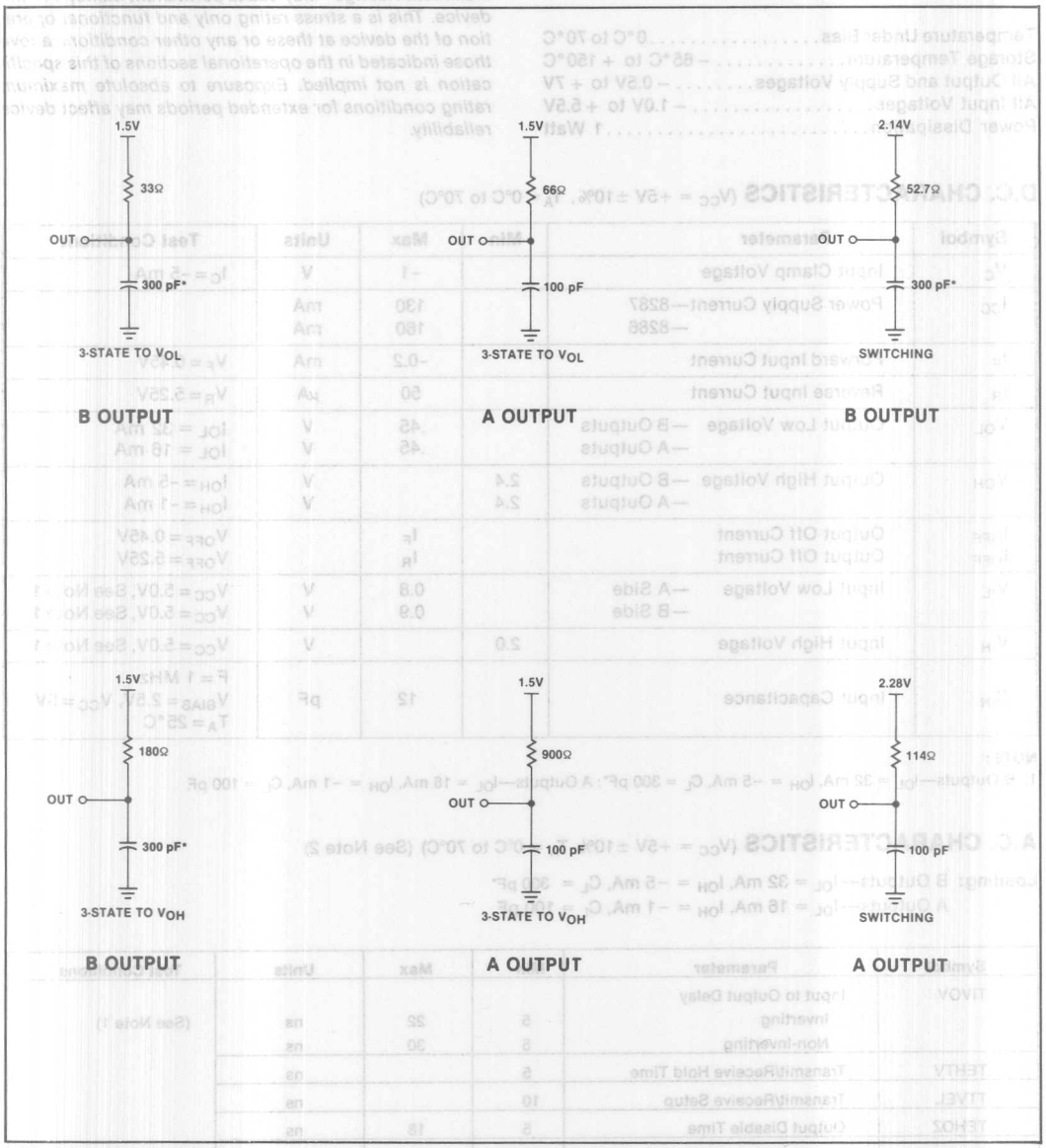
FUNCTIONAL DESCRIPTION

The 8286 and 8287 transceivers are 8-bit transceivers with high impedance outputs. With T active HIGH and $\overline{OE}$ active LOW, data at the A_0–A_7 pins is driven onto the B_0–B_7 pins. With T inactive LOW and $\overline{OE}$ active LOW, data at the B_0–B_7 pins is driven onto the A_0–A_7 pins. No output low glitching will occur whenever the transceivers are entering or leaving the high impedance state.



AFN-01506D

TEST LOAD CIRCUITS



*200 pF for plastic 8286/8287

AFN-01506D

ABSOLUTE MAXIMUM RATINGS*

Temperature Under Bias 0°C to 70°C
Storage Temperature −65°C to +150°C
All Output and Supply Voltages −0.5V to +7V
All Input Voltages −1.0V to +5.5V
Power Dissipation 1 Watt

NOTICE: Stresses above those listed under "Absolute Maximum Ratings" may cause permanent damage to the device. This is a stress rating only and functional operation of the device at these or any other conditions above those indicated in the operational sections of this specification is not implied. Exposure to absolute maximum rating conditions for extended periods may affect device reliability.

D.C. CHARACTERISTICS (V_{CC} = +5V ±10%, T_A = 0°C to 70°C)

Symbol	Parameter	Min	Max	Units	Test Conditions
V_C	Input Clamp Voltage		-1	V	I_C = -5 mA
I_{CC}	Power Supply Current—8287 —8286		130 160	mA mA	
I_F	Forward Input Current		-0.2	mA	V_F = 0.45V
I_R	Reverse Input Current		50	μA	V_R = 5.25V
V_{OL}	Output Low Voltage —B Outputs —A Outputs		.45 .45	V V	I_{OL} = 32 mA I_{OL} = 16 mA
V_{OH}	Output High Voltage —B Outputs —A Outputs	2.4 2.4		V V	I_{OH} = -5 mA I_{OH} = -1 mA
I_{OFF} I_{OFF}	Output Off Current Output Off Current		I_F I_R		V_{OFF} = 0.45V V_{OFF} = 5.25V
V_{IL}	Input Low Voltage —A Side —B Side		0.8 0.9	V V	V_{CC} = 5.0V, See Note 1 V_{CC} = 5.0V, See Note 1
V_{IH}	Input High Voltage	2.0		V	V_{CC} = 5.0V, See Note 1
C_{IN}	Input Capacitance		12	pF	F = 1 MHz V_{BIAS} = 2.5V, V_{CC} = 5V T_A = 25°C

NOTE:
1. B Outputs—I_{OL} = 32 mA, I_{OH} = −5 mA, C_L = 300 pF*: A Outputs—I_{OL} = 16 mA, I_{OH} = −1 mA, C_L = 100 pF.

A.C. CHARACTERISTICS (V_{CC} = +5V ±10%, T_A = 0°C to 70°C) (See Note 2)

Loading: B Outputs—I_{OL} = 32 mA, I_{OH} = −5 mA, C_L = 300 pF*
A Outputs—I_{OL} = 16 mA, I_{OH} = −1 mA, C_L = 100 pF

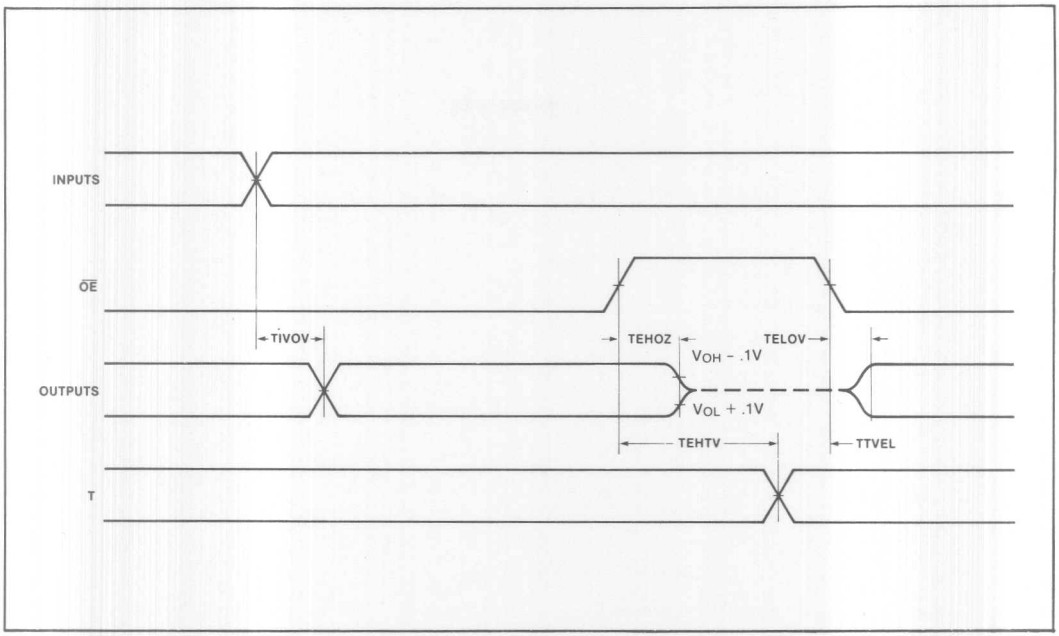
Symbol	Parameter	Min	Max	Units	Test Conditions
TIVOV	Input to Output Delay Inverting Non-Inverting	 5 5	 22 30	 ns ns	(See Note 1)
TEHTV	Transmit/Receive Hold Time	5		ns	
TTVEL	Transmit/Receive Setup	10		ns	
TEHOZ	Output Disable Time	5	18	ns	
TELOV	Output Enable Time	10	30	ns	
TOLOH	Input, Output Rise Time		20	ns	From 0.8 V to 2.0V
TOHOL	Input, Output Fall Time		12	ns	From 2.0V to 8.0V

*C_L = 200 pF for plastic 8286/8287

NOTE:
1. See waveforms and test load circuit on following page.
2. For Extended Temperature EXPRESS the Preliminary Maximum Values are TIVOV = 25 vs 22, 35 vs 30; TEHOZ = 25; TELOV = 50.

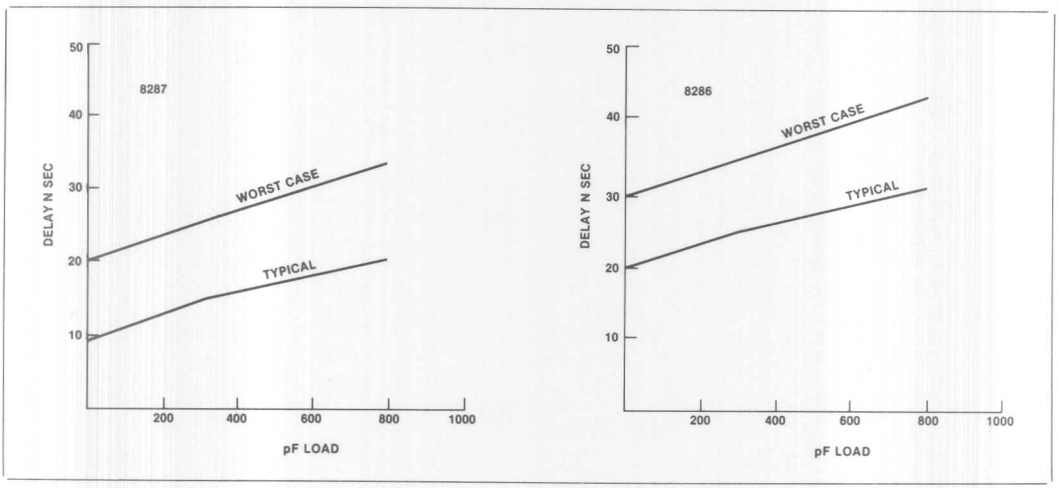
AFN-01506D

WAVEFORMS



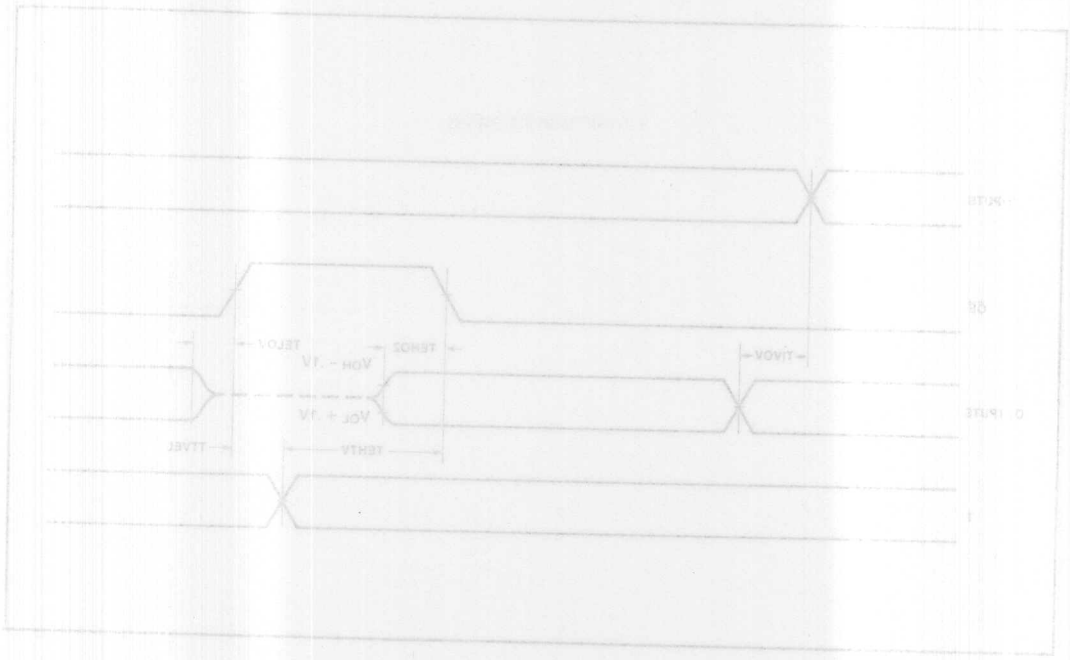
NOTE:

1. All timing measurements are made at 1.5V unless otherwise noted.



Output Delay versus Capacitance

AFN-01506D



8207
ADVANCED DYNAMIC RAM CONTROLLER

- Provides All Signals Necessary to Control 16K (2118), 64K (2164A) and 256K Dynamic RAMs
- Directly Addresses and Drives up to 2 Megabytes without External Drivers
- Supports Single and Dual-Port Configurations
- Automatic RAM Initialization in All Modes
- Five Programmable Refresh Modes
- Transparent Memory Scrubbing in ECC Mode
- Supports Intel iAPX 86, 88, 186, and 286 Microprocessors
- Data Transfer Acknowledge Signals for Each Port
- Provides Signals to Directly Control the 8206 Error Detection and Correction Unit
- Supports Synchronous or Asynchronous Operation on Either Port
- +5 Volt Only HMOSII Technology for High Performance and Low Power

The Intel 8207 Advanced Dynamic RAM Controller (ADRC) is a high-performance, systems-oriented, Dynamic RAM controller that is designed to easily interface 16K, 64K and 256K Dynamic RAMs to Intel and other microprocessor Systems. A dual-port interface allows two different busses to independently access memory. When configured with an 8206 Error Detection and Correction Unit the 8207 supplies the necessary logic for designing large error-corrected memory arrays. This combination provides automatic memory initialization and transparent memory error scrubbing.

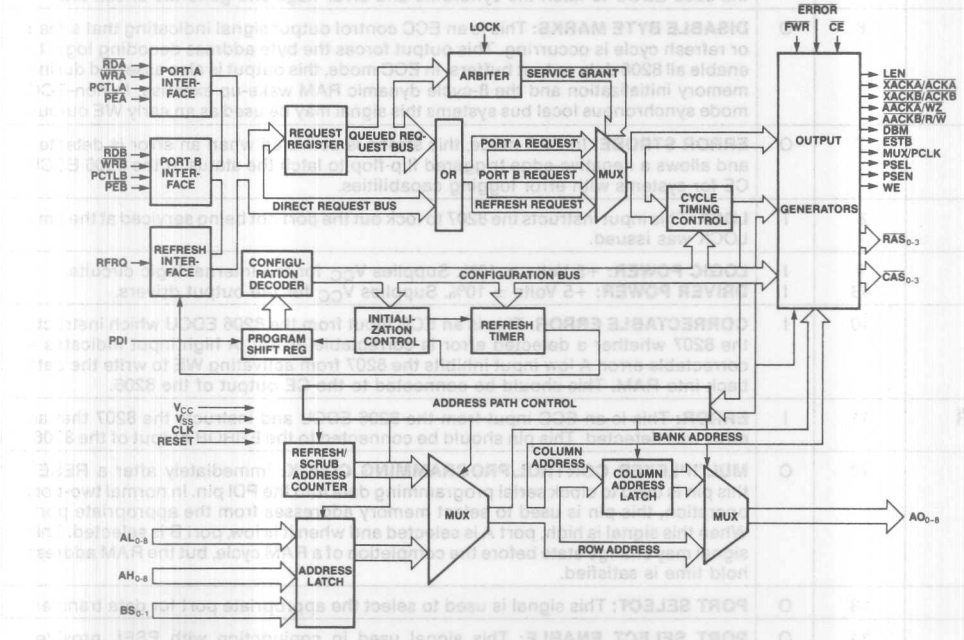


Figure 1. 8207 Block Diagram

Intel Corporation Assumes No Responsibility for the Use of Any Circuitry Other Than Circuitry Embodied in an Intel Product. No Other Circuit Patent Licenses are Implied. Information Contained Herein Supercedes Previously Published Specifications On These Devices From Intel.
© INTEL CORPORATION, 1983.

NOVEMBER 1982
ORDER NUMBER: 210463-002

Table 1. Pin Description

Symbol	Pin	Type	Name and Function
LEN	1	O	**ADDRESS LATCH ENABLE:** In two-port configurations, when port A is running with iAPX 286 Status interface mode, this output replaces the ALE signal from the system bus controller and generates an address latch enable signal which provides optimum setup and hold timing for the 8207.
$\overline{XACKA}$/ $\overline{ACKA}$	2	O	**TRANSFER ACKNOWLEDGE PORT A/ACKNOWLEDGE PORT A:** In non-ECC mode, this pin is $\overline{XACKA}$ and indicates that data on the bus is valid during a read cycle or that data may be removed from the bus during a write cycle for Port A. $\overline{XACKA}$ is a Multibus-compatible signal. In ECC mode, this pin is $\overline{ACKA}$ which can be configured, depending on the programming of the X program bit, as an XACK or AACK strobe. The SA programming bit determines whether AACK will be early or late.
$\overline{XACKB}$/ $\overline{ACKB}$	3	O	**TRANSFER ACKNOWLEDGE PORT B/ACKNOWLEDGE PORT B:** In non-ECC mode, this pin is $\overline{XACKB}$ and indicates that data on the bus is valid during a read cycle or that data may be removed from the bus during a write cycle for Port B. $\overline{XACKB}$ is a Multibus-compatible signal. In ECC mode, this pin is $\overline{ACKB}$ which can be configured, depending on the programming of the X program bit, as an XACK or AACK strobe. The SB programming bit determines whether AACK will be early or late.
$\overline{AACKA}$/ $\overline{WZ}$	4	O	**ADVANCED ACKNOWLEDGE PORT A/WRITE ZERO:** In non-ECC mode, this pin is $\overline{AACKA}$ and indicates that the processor may continue processing and that data will be available when required. This signal is optimized for the system by programming the SA program bit for synchronous or asynchronous operation. After a RESET, this signal will cause the 8206 to force the data to all zeros and generate the appropriate check bits.
$\overline{AACKB}$/ R/$\overline{W}$	5	O	**ADVANCED ACKNOWLEDGE PORT B/READ/WRITE:** In non-ECC mode, this pin is $\overline{AACKB}$ and indicates that the processor may continue processing and that data will be available when required. This signal is optimized for the system by programming the SB program bit for synchronous or asynchronous operation. This signal causes the 8206 EDCU to latch the syndrome and error flags and generate check bits.
$\overline{DBM}$	6	O	**DISABLE BYTE MARKS:** This is an ECC control output signal indicating that a read or refresh cycle is occurring. This output forces the byte address decoding logic to enable all 8206 data output buffers. In ECC mode, this output is also asserted during memory initialization and the 8-cycle dynamic RAM wake-up exercise. In non-ECC mode synchronous local bus systems this signal may be used as an early WE output.
$\overline{ESTB}$	7	O	**ERROR STROBE:** In ECC mode, this strobe is activated when an error is detected and allows a negative-edge triggered flip-flop to latch the status of the 8206 EDCU CE for systems with error logging capabilities.
LOCK	8	I	**LOCK:** This input instructs the 8207 to lock out the port not being serviced at the time LOCK was issued.
V_{CC}	9 43	I I	**LOGIC POWER:** +5 Volts ± 10%. Supplies V_{CC} for the internal logic circuits. **DRIVER POWER:** +5 Volts ± 10%. Supplies V_{CC} for the output drivers.
CE	10	I	**CORRECTABLE ERROR:** This is an ECC input from the 8206 EDCU which instructs the 8207 whether a detected error is correctable or not. A high input indicates a correctable error. A low input inhibits the 8207 from activating WE to write the data back into RAM. This should be connected to the CE output of the 8206.
$\overline{ERROR}$	11	I	**ERROR:** This is an ECC input from the 8206 EDCU and instructs the 8207 that an error was detected. This pin should be connected to the ERROR output of the 8206.
MUX/ PCLK	12	O	**MULTIPLEXER CONTROL/PROGRAMMING CLOCK:** Immediately after a RESET this pin is used to clock serial programming data into the PDI pin. In normal two-port operation, this pin is used to select memory addresses from the appropriate port. When this signal is high, port A is selected and when it is low, port B is selected. This signal may change state before the completion of a RAM cycle, but the RAM address hold time is satisfied.
PSEL	13	O	**PORT SELECT:** This signal is used to select the appropriate port for data transfer.
PSEN	14	O	**PORT SELECT ENABLE:** This signal used in conjunction with PSEL provides contention-free port exchange. When PSEN is low, PSEL is allowed to change state.
WE	15	O	**WRITE ENABLE:** This signal provides the dynamic RAM array the write enable input for a write operation.

AFN-02218B

Table 1. Pin Description (Continued)

Symbol	Pin	Type	Name and Function
$\overline{FWR}$	16	I	**FULL WRITE:** This is an ECC input signal that instructs the 8207, in an ECC configuration, whether the present write cycle is normal RAM write (full write) or a RAM partial write (read-modify-write) cycle.
RESET	17	I	**RESET:** This signal causes all internal counters and state flip-flops to be reset and upon release of RESET, data appearing at the PDI pin is clocked in by the PCLK output. The states of the PDI, PCTLA, PCTLB and RFRQ pins are sampled by RESET going inactive and are used to program the 8207.
$\overline{CAS0}$	18	O	**COLUMN ADDRESS STROBE:** These outputs are used by the dynamic RAM array to latch the column address, present on the AO0–8 pins. These outputs are selected by the BS0 and BS1 as programmed by program bits RB0 and RB1. These outputs drive the dynamic RAM array directly and need no external drivers.
$\overline{CAS1}$	19	O	
$\overline{CAS2}$	20	O	
$\overline{CAS3}$	21	O	
$\overline{RAS0}$	22	O	**ROW ADDRESS STROBE:** These outputs are used by the dynamic RAM array to latch the row address, present on the AO0–8 pins. These ouptuts are selected by the BS0 and BS1 as programmed by program bits RB0 and RB1. These outputs drive the dynamic RAM array directly and need no external drivers.
$\overline{RAS1}$	23	O	
$\overline{RAS2}$	24	O	
$\overline{RAS3}$	25	O	
V_{SS}	26	I	**DRIVER GROUND:** Provides a ground for the output drivers.
	60	I	**LOGIC GROUND:** Provides a ground for the remainder of the device.
AO0	35	O	**ADDRESS OUTPUTS:** These outputs are designed to provide the row and column addresses of the selected port to the dynamic RAM array. These outputs drive the dynamic RAM array directly and need no external drivers.
AO1	34	O	
AO2	33	O	
AO3	32	O	
AO4	31	O	
AO5	30	O	
AO6	29	O	
AO7	28	O	
AO8	27	O	
BS0	36	I	**BANK SELECT:** These inputs are used to select one of four banks of the dynamic RAM array as defined by the program bits RB0 and RB1.
BS1	37	I	
AL0	38	I	**ADDRESS LOW:** These lower-order address inputs are used to generate the row address for the internal address multiplexer.
AL1	39	I	
AL2	40	I	
AL3	41	I	
AL4	42	I	
AL5	44	I	
AL6	45	I	
AL7	46	I	
AL8	47	I	
AH0	48	I	**ADDRESS HIGH:** These higher-order address inputs are used to generate the column address for the internal address multiplexer.
AH1	49	I	
AH2	50	I	
AH3	51	I	
AH4	52	I	
AH5	53	I	
AH6	54	I	
AH7	55	I	
AH8	56	I	
PDI	57	I	**PROGRAM DATA INPUT:** This input programs the various user-selectable options in the 8207. The PCLK pin shifts programming data into the PDI input from optional external shift registers. This pin may be strapped high or low to a default ECC (PDI = V_{CC}) or non-ECC (PDI = Ground) mode configuration.
RFRQ	58	I	**REFRESH REQUEST:** This input is sampled on the falling edge of RESET. If it is high at RESET, then the 8207 is programmed for internal refresh request or external refresh request with failsafe protection. If it is low at RESET, then the 8207 is programmed for external refresh without failsafe protection or burst refresh. Once programmed the RFRQ pin accepts signals to start an external refresh with failsafe protection or external refresh without failsafe protection or a burst refresh.

AFN-02218B

Table 1. Pin Description (Continued)

Symbol	Pin	Type	Name and Function
CLK	59	I	**CLOCK:** This input provides the basic timing for sequencing the internal logic.
$\overline{RDB}$	61	I	**READ FOR PORT B:** This pin is the read memory request command input for port B. This input also directly accepts the $\overline{S1}$ status line from Intel processors.
$\overline{WRB}$	62	I	**WRITE FOR PORT B:** This pin is the write memory request command input for port B. This input also directly accepts the $\overline{S0}$ status line from Intel processors.
$\overline{PEB}$	63	I	**PORT ENABLE FOR PORT B:** This pin serves to enable a RAM cycle request for port B. It is generally decoded from the port address.
PCTLB	64	I	**PORT CONTROL FOR PORT B:** This pin is sampled on the falling edge of RESET. It configures port B to accept command inputs or processor status inputs. If low after RESET, the 8207 is programmed to accept command or iAPX 286 status inputs or Multibus commands. If high after RESET, the 8207 is programmed to accept status inputs from iAPX 86 or iAPX 186 processors. The $\overline{S2}$ status line should be connected to this input if programmed to accept iAPX 86 or iAPX 186 status inputs. When programmed to accept commands or iAPX 286 status, it should be tied low or it may be used as a Multibus-compatible inhibit signal.
$\overline{RDA}$	65	I	**READ FOR PORT A:** This pin is the read memory request command input for port A. This input also directly accepts the $\overline{S1}$ status line from Intel processors.
$\overline{WRA}$	66	I	**WRITE FOR PORT A:** This pin is the write memory request command input for port A. This input also directly accepts the $\overline{S0}$ status line from Intel processors.
$\overline{PEA}$	67	I	**PORT ENABLE FOR PORT A:** This pin serves to enable a RAM cycle request for port A. It is generally decoded from the port address.
PCTLA	68	I	**PORT CONTROL FOR PORT A:** This pin is sampled on the falling edge of RESET. It configures port A to accept command inputs or processor status inputs. If low after RESET, the 8207 is programmed to accept command or iAPX 286 status inputs or Multibus commands. If high after RESET, the 8207 is programmed to accept status inputs from iAPX 86 or iAPX 186 processors. The $\overline{S2}$ status line should be connected to this input if programmed to accept iAPX 86 or iAPX 186 status inputs. When programmed to accept commands or iAPX 286 status, it should be tied low or it may be connected to INHIBIT when operating with Multibus.

GENERAL DESCRIPTION

The Intel 8207 Advanced Dynamic RAM Controller (ADRC) is a microcomputer peripheral device which provides the necessary signals to address, refresh and directly drive 16K, 64K and 256K dynamic RAMs. This controller also provides the necessary arbitration circuitry to support dual-port access of the dynamic RAM array.

The ADRC supports several microprocessor interface options including synchronous and asynchronous connection to iAPX 86, iAPX 88, iAPX 186, iAPX 286 and Multibus.

This device may be used with the 8206 Error Detection and Correction Unit (EDCU). When used with the 8206, the 8207 is programmed in the Error Checking and Correction (ECC) mode. In this mode, the 8207 provides all the necessary control signals for the 8206 to perform memory initialization and transparent error scrubbing during refresh.

FUNCTIONAL DESCRIPTION

Processor Interface

The 8207 has control circuitry for two ports each capable of supporting one of several possible bus structures. The ports are independently configurable allowing the dynamic RAM to serve as an interface between two different bus structures.

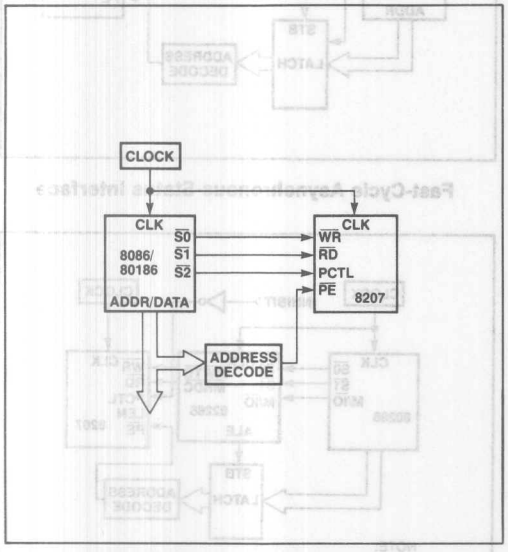
Each port of the 8207 may be programmed to run synchronous or asynchronous to the processor clock. (See Synchronous/Asynchronous Mode) The 8207 has been optimized to run synchronously with Intel's iAPX 86, iAPX 88, iAPX 186 and iAPX 286. When the 8207 is programmed to run in asynchronous mode, the 8207 inserts the necessary synchronization circuitry for the $\overline{RD}$, $\overline{WR}$, $\overline{PE}$, and PCTL inputs.

AFN-02218B

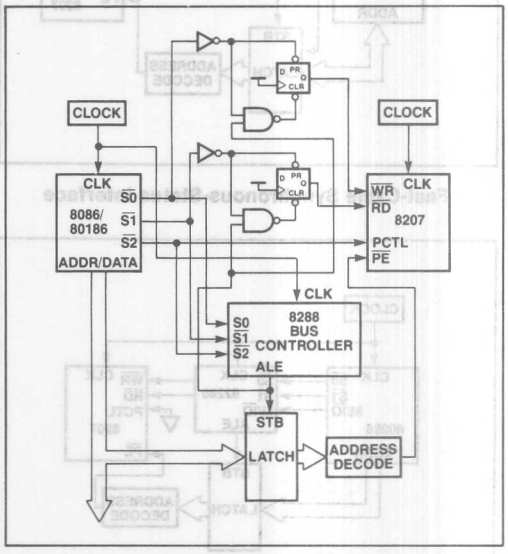
The 8207 can also decode the status lines directly from the iAPX 86, iAPX 88, iAPX 186 and the iAPX 286 or can be programmed to receive read or write Multibus commands or commands from a bus controller. (See Status/Command Mode)

The 8207 may be programmed to accept the clock of the iAPX 86, 88, 186, or 286. The 8207 adjusts its internal timing to allow for the different clock frequencies of these microprocessors. (See Microprocessor Clock Frequency Option)

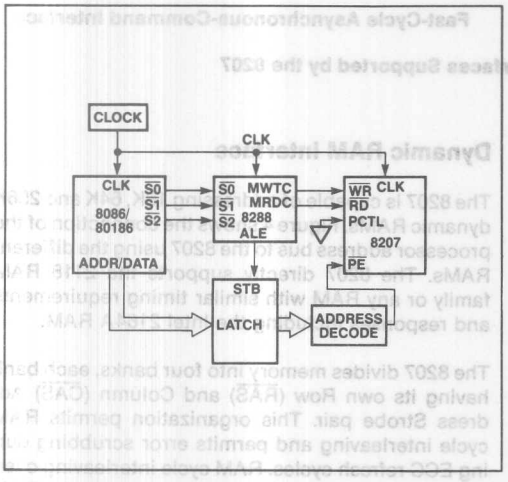
Figure 2 shows the different processor interfaces to the 8207 using the synchronous or asynchronous mode and status or command interface.



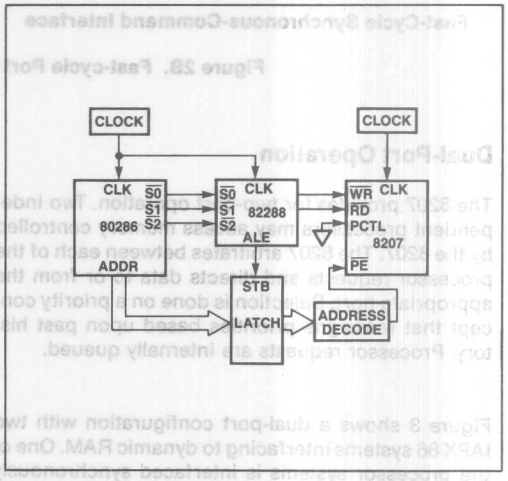
Slow-Cycle Synchronous-Status Interface



Slow-Cycle Asynchronous-Status Interface



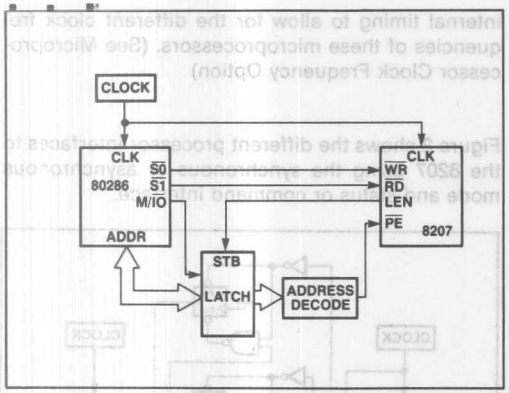
Slow-Cycle Synchronous-Command Interface



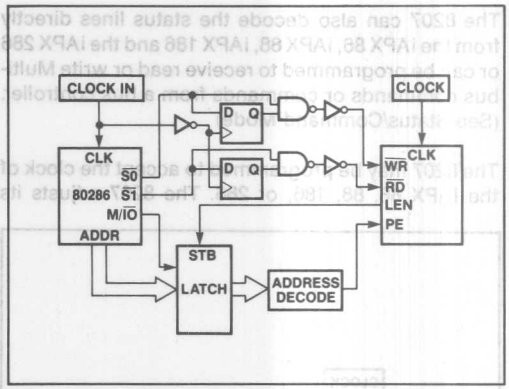
Slow-Cycle Asynchronous-Command Interface

Figure 2A. Slow-cycle Port Interfaces Supported by the 8207

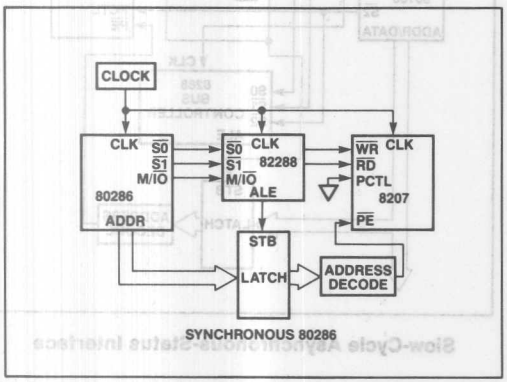
AFN-02218B



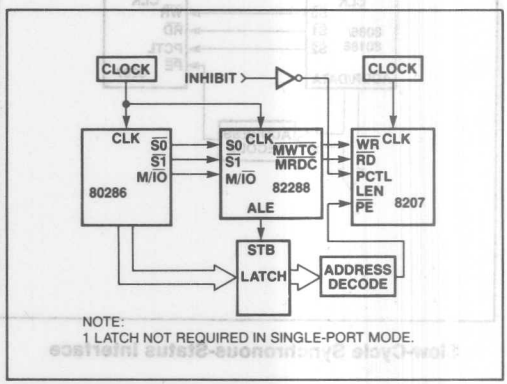
Fast-Cycle Synchronous-Status Interface



Fast-Cycle Asynchronous-Status Interface



Fast-Cycle Synchronous-Command Interface



Fast-Cycle Asynchronous-Command Interface

Figure 2B. Fast-cycle Port Interfaces Supported by the 8207

Dual-Port Operation

The 8207 provides for two-port operation. Two independent processors may access memory controlled by the 8207. The 8207 arbitrates between each of the processor requests and directs data to or from the appropriate port. Selection is done on a priority concept that reassigns priorities based upon past history. Processor requests are internally queued.

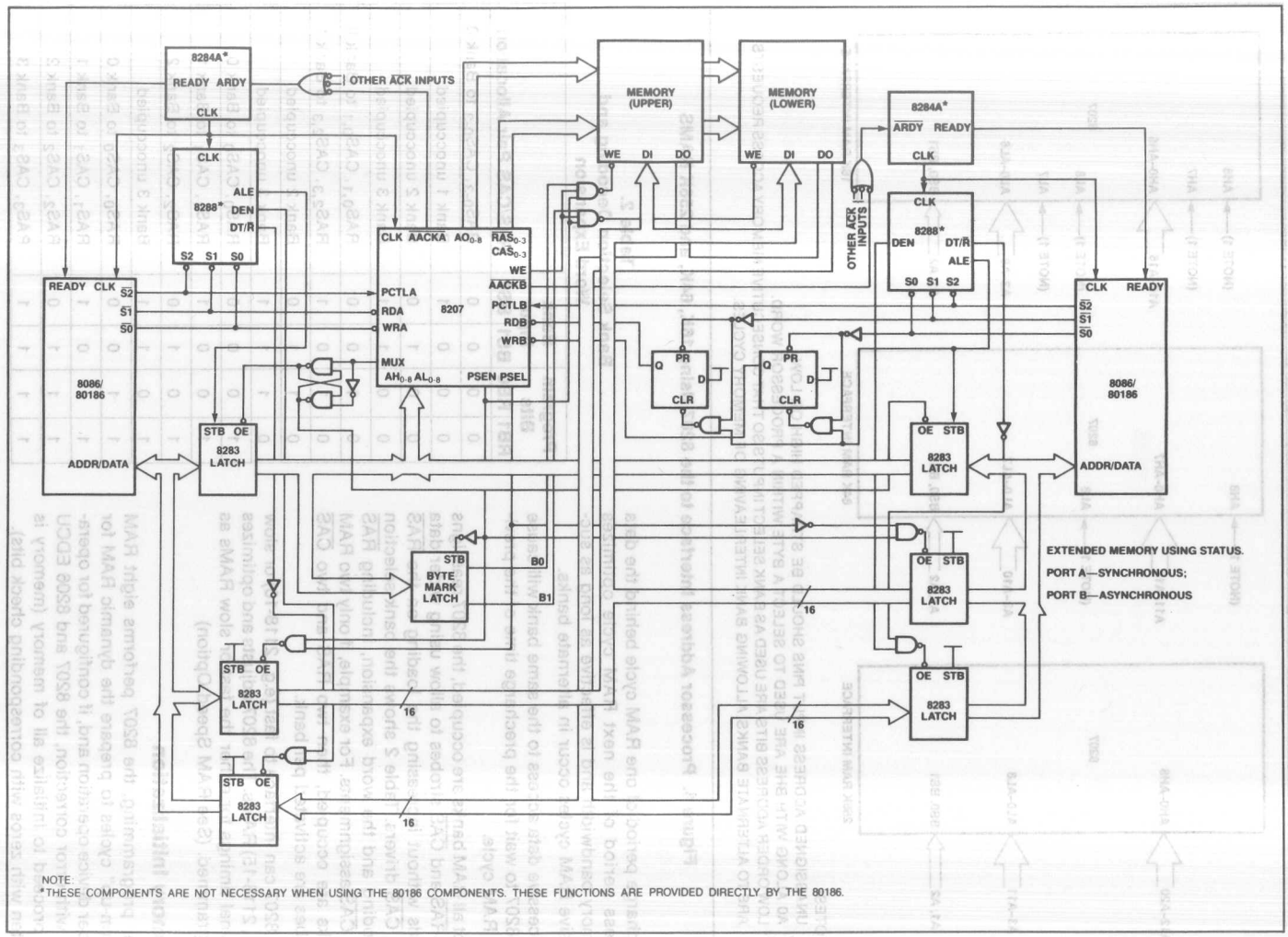
Figure 3 shows a dual-port configuration with two iAPX 86 systems interfacing to dynamic RAM. One of the processor systems is interfaced synchronously using the status interface and the other is interfaced asynchronously also using the status interface.

Dynamic RAM Interface

The 8207 is capable of addressing 16K, 64K and 256K dynamic RAMs. Figure 4 shows the connection of the processor address bus to the 8207 using the different RAMs. The 8207 directly supports the 2118 RAM family or any RAM with similar timing requirements and responses including the Intel 2164A RAM.

The 8207 divides memory into four banks, each bank having its own Row ($\overline{RAS}$) and Column ($\overline{CAS}$) Address Strobe pair. This organization permits RAM cycle interleaving and permits error scrubbing during ECC refresh cycles. RAM cycle interleaving overlaps the start of the next RAM cycle with the RAM Precharge period of the previous cycle. Hiding the

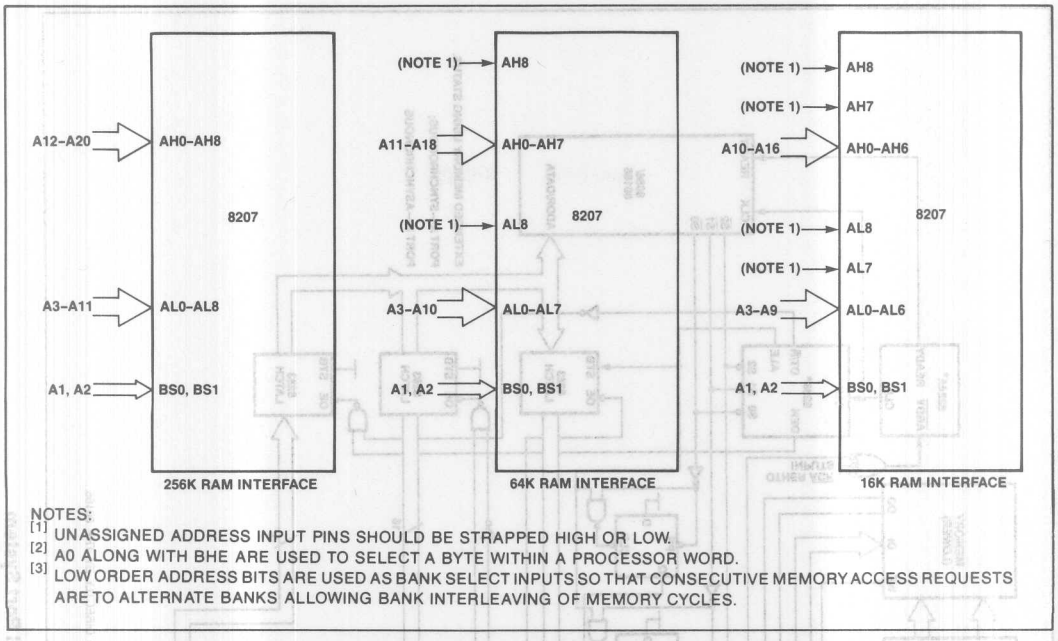
AFN-02218B



NOTE:
*THESE COMPONENTS ARE NOT NECESSARY WHEN USING THE 80186 COMPONENTS. THESE FUNCTIONS ARE PROVIDED DIRECTLY BY THE 80186.

Figure 3. 8086/80186 Dual Port System

AFN-02218B



NOTES:
[1] UNASSIGNED ADDRESS INPUT PINS SHOULD BE STRAPPED HIGH OR LOW.
[2] A0 ALONG WITH BHE ARE USED TO SELECT A BYTE WITHIN A PROCESSOR WORD.
[3] LOW ORDER ADDRESS BITS ARE USED AS BANK SELECT INPUTS SO THAT CONSECUTIVE MEMORY ACCESS REQUESTS ARE TO ALTERNATE BANKS ALLOWING BANK INTERLEAVING OF MEMORY CYCLES.

Figure 4. Processor Address Interface to the 8207 Using 16K, 64K, and 256K RAMS

precharge period of one RAM cycle behind the data access period of the next RAM cycle optimizes memory bandwidth and is effective as long as successive RAM cycles occur in alternate banks.

Successive data access to the same bank will cause the 8207 to wait for the precharge time of the previous RAM cycle.

If not all RAM banks are occupied, the 8207 reassigns the $\overline{RAS}$ and $\overline{CAS}$ strobes to allow using wider data words without increasing the loading on the $\overline{RAS}$ and $\overline{CAS}$ drivers. Table 2 shows the bank selection decoding and the word expansion, including $\overline{RAS}$ and $\overline{CAS}$ assignments. For example, if only two RAM banks are occupied, then two $\overline{RAS}$ and two $\overline{CAS}$ strobes are activated per bank.

The 8207 can interface to fast (e.g., 2118-10) or slow (e.g., 2118-15) RAMs. The 8207 adjusts and optimizes internal timings for either the fast or slow RAMs as programmed. (See RAM Speed Option)

Memory Initialization

After programming, the 8207 performs eight RAM "warm-up" cycles to prepare the dynamic RAM for proper device operation and, if configured for operation with error correction, the 8207 and 8206 EDCU will proceed to initialize all of memory (memory is written with zeros with corresponding check bits).

Table 2.
Bank Selection Decoding and Word Expansion

Program Bits		Bank Input		
RB1	RB0	BS1	BSØ	RAS/CAS Pair Allocation
0	0	0	0	RAS_{0-3}, CAS_{0-3} to Bank 0
0	0	0	1	Bank 1 unoccupied
0	0	1	0	Bank 2 unoccupied
0	0	1	1	Bank 3 unoccupied
0	1	0	0	$RAS_{0,1}$, $CAS_{0,1}$ to Bank 0
0	1	0	1	$RAS_{2,3}$, $CAS_{2,3}$ to Bank 1
0	1	1	0	Bank 2 unoccupied
0	1	1	1	Bank 3 unoccupied
1	0	0	0	RAS_0, CAS_0 to Bank 0
1	0	0	1	RAS_1, CAS_1 to Bank 1
1	0	1	0	RAS_2, CAS_2 to Bank 2
1	0	1	1	Bank 3 unoccupied
1	1	0	0	RAS_0, CAS_0 to Bank 0
1	1	0	1	RAS_1, CAS_1 to Bank 1
1	1	1	0	RAS_2, CAS_2 to Bank 2
1	1	1	1	RAS_3, CAS_3 to Bank 3

AFN-02218B

Because the time to initialize memory is fairly long, the 8207 may be programmed to skip initialization in ECC mode. The time required to initialize all of memory is dependent on the clock cycle time to the 8207 and can be calculated by the following equation:

eq.1 $T_{INIT} = (2^{23})\ T_{CY}$

if T_{CY} = 125 ns then $T_{INIT} \approx 1$ sec.

8206 ECC Interface

For operation with Error Checking and Correction (ECC), the 8207 adjusts its internal timing and changes some pin functions to optimize performance and provide a clean dual-port memory interface between the 8206 EDCU and memory. The 8207 directly supports a master-only (16-bit word plus 6 check bits) system. Under extended operation and reduced clock frequency, the 8207 will support any ECC master-slave configuration up to 80 data bits, which is the maximum set by the 8206 EDCU. (See Extend Option)

Correctable errors detected during memory read cycles are corrected immediately and then written back into memory.

In a synchronous bus environment, ECC system performance has been optimized to enhance processor throughput, while in an asynchronous bus environment (the Multibus), ECC performance has been optimized to get valid data onto the bus as quickly as possible. Performance optimization, processor throughput or quick data access may be selected via the Transfer Acknowledge Option.

The main difference between the two ECC implementations is that, when optimized for processor throughput, RAM data is always corrected and an advanced transfer acknowledge is issued at a point when, by knowing the processor characteristics, data is guaranteed to be valid by the time the processor needs it.

When optimized for quick data access, (valid for Multibus) the 8206 is configured in the uncorrecting mode where the delay associated with error correction circuitry is transparent, and a transfer acknowledge is issued as soon as valid data is known to exist. If the $\overline{\text{ERROR}}$ flag is activated, then the transfer acknowledge is delayed until after the 8207 has instructed the 8206 to correct the data and the corrected data becomes available on the bus. Figure 5 illustrates a dual-port ECC system.

Figure 6 illustrates the interface required to drive the $\overline{\text{CRCT}}$ pin of the 8206, in the case that one port (PORT A) receives an advanced acknowledge (not Multibus-compatible), while the other port (PORT B) receives $\overline{\text{XACK}}$ (which is Multibus-compatible).

Error Scrubbing

The 8207/8206 performs error correction during refresh cycles (error scrubbing). Since the 8207 must refresh RAM, performing error scrubbing during refresh allows it to be accomplished without additional performance penalties.

Upon detection of a correctable error during refresh, the RAM refresh cycle is lengthened slightly to permit the 8206 to correct the error and for the corrected word to be rewritten into memory. Uncorrectable errors detected during scrubbing are ignored.

Refresh

The 8207 provides an internal refresh interval counter and a refresh address counter to allow the 8207 to refresh memory. The 8207 will refresh 128 rows every 2 milliseconds or 256 rows every 4 milliseconds, which allows all RAM refresh options to be supported. In addition, there exists the ability to refresh 256 row address locations every 2 milliseconds via the Refresh Period programming option.

The 8207 may be programmed for any of five different refresh options: Internal refresh only, External refresh with failsafe protection, External refresh without failsafe protection, Burst Refresh mode, or no refresh. (See Refresh Options)

It is possible to decrease the refresh time interval by 10%, 20% or 30%. This option allows the 8207 to compensate for reduced clock frequencies. Note that an additional 5% interval shortening is built-in in all refresh interval options to compensate for clock variations and non-immediate response to the internally generated refresh request. (See Refresh Period Options)

External Refresh Requests after RESET

External refresh requests are not recognized by the 8207 until after it is finished programming and preparing memory for access. Memory preparation includes 8 RAM cycles to prepare and ensure proper

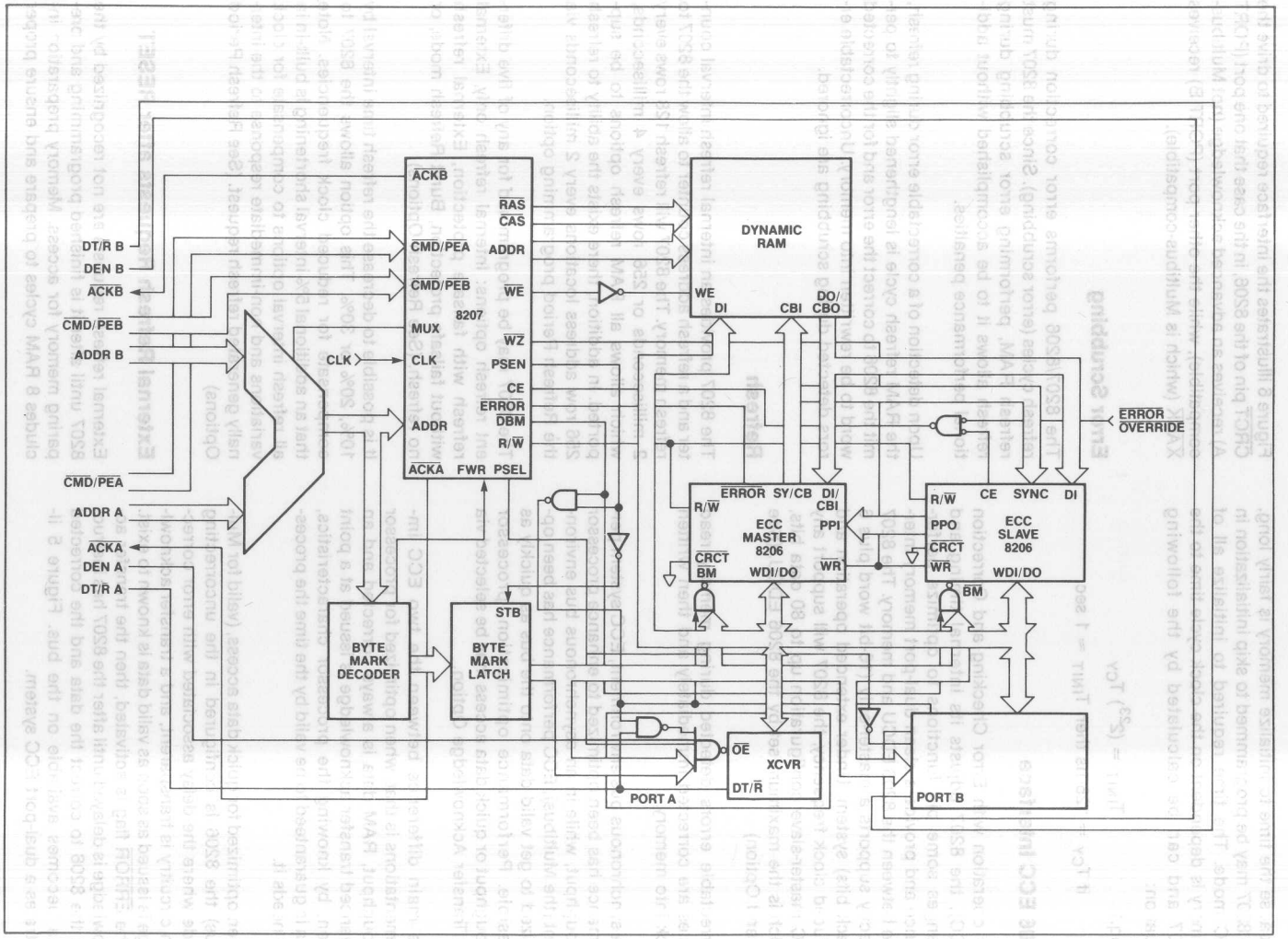


Figure 5. Two-Port ECC Implementation Using the 8207 and the 8206

AFN-02218B

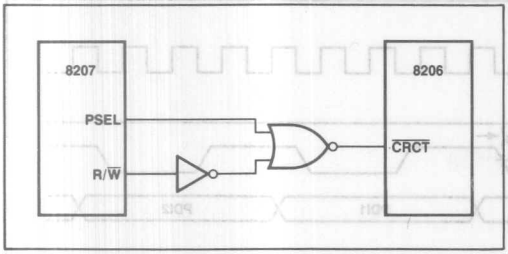


Figure 6. Interface to 8206 $\overline{\text{CRCT}}$ Input When Port A Receives $\overline{\text{AACK}}$ and Port B Receives $\overline{\text{XACK}}$

dynamic RAM operation, and memory initialization if error correction is used. Many dynamic RAMs require this warm-up period for proper operation. The time it takes for the 8207 to recognize a request is shown below.

eq. 2 Non-ECC Systems: $T_{RESP} = T_{PROG} + T_{PREP}$

eq. 3 where: $T_{PROG} = (66)\ (T_{CY})$ which is programming time

eq. 4 $T_{PREP} = (8)\ (32)\ (T_{CY})$ which is the RAM warm-up time

if $T_{CY} = 125$ ns then $T_{RESP} \approx 41$ us

eq. 5 ECC Systems: $T_{RESP} = T_{PROG} + T_{PREP} + T_{INIT}$

if $T_{CY} = 125$ ns then $T_{RESP} \approx 1$ sec

RESET

RESET is an asynchronous input, the falling edge of which is used by the 20 to directly sample the logic levels of the PCTLA, PCTLB, RFRQ, and PDI inputs. The internally synchronized falling edge of RESET is used to begin programming operations (shifting in the contents of the external shift register into the PDI input).

Until programming is complete the 8207 registers but does not respond to command or status inputs. A simple means of preventing commands or status from occurring during this period is to differentiate the system reset pulse to obtain a smaller reset pulse for the 8207. The total time of the reset pulse and the 8207 programming time must be less than the time before the first command in systems that alter the default port synchronization programming bits (default is Port A synchronous, Port B asynchronous). Differentiated reset is unnecessary when the default port synchronization programming is used. The differentiated reset pulse would be shorter than the system reset pulse by at least the programming period required by the 8207. The differentiated reset pulse first resets the 8207, and system reset would reset the rest of the system. While the rest of the system is still in reset, the 8207 completes its programming. Figure 7 illustrates a circuit to accomplish this task.

Within four clocks after RESET goes active, all the 8207 outputs will go high, except for PSEN, WE, and AO0-8, which will go low.

OPERATIONAL DESCRIPTION

Programming the 8207

The 8207 is programmed after reset. On the falling edge of RESET, the logic states of several input pins are latched internally. The falling edge of RESET actually performs the latching, which means that the logic levels on these inputs must be stable prior to that time. The inputs whose logic levels are latched at the end of reset are the PCTLA, PCTLB, REFRQ, and PDI pins. Figure 8 shows the necessary timing for programming the 8207.

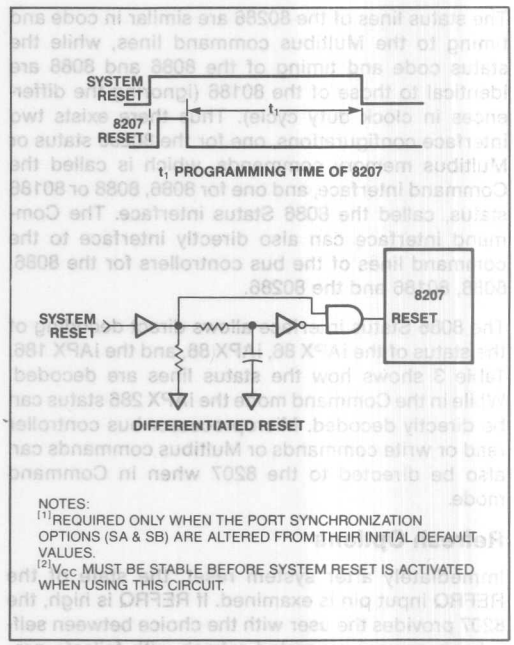
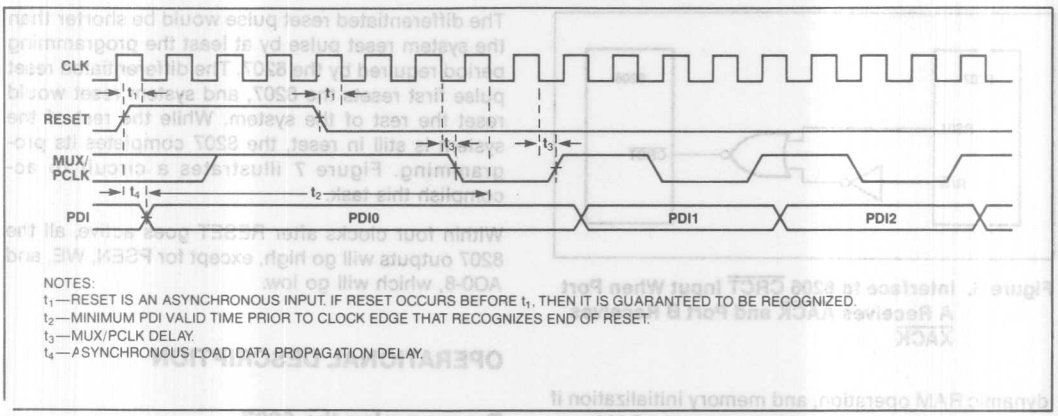


Figure 7. 8207 Differentiated Reset Circuit

AFN-02218B



NOTES:
t_1—RESET IS AN ASYNCHRONOUS INPUT. IF RESET OCCURS BEFORE t_1, THEN IT IS GUARANTEED TO BE RECOGNIZED.
t_2—MINIMUM PDI VALID TIME PRIOR TO CLOCK EDGE THAT RECOGNIZES END OF RESET.
t_3—MUX/PCLK DELAY.
t_4—ASYNCHRONOUS LOAD DATA PROPAGATION DELAY.

Figure 8. Timing Illustrating External Shift Register Requirements for Programming the 8207

Status/Command Mode

The two processor ports of the 8207 are configured by the states of the PCTLA and PCTLB pins. Which interface is selected depends on the state of the individual port's PCTL pin at the end of reset. If PCTL is high at the end of the reset, the 8086 Status interface is selected; if it is low, then the Command interface is selected.

The status lines of the 80286 are similar in code and timing to the Multibus command lines, while the status code and timing of the 8086 and 8088 are identical to those of the 80186 (ignoring the differences in clock duty cycle). Thus there exists two interface configurations, one for the 80286 status or Multibus memory commands, which is called the Command interface, and one for 8086, 8088 or 80186 status, called the 8086 Status interface. The Command interface can also directly interface to the command lines of the bus controllers for the 8086, 8088, 80186 and the 80286.

The 8086 Status interface allows direct decoding of the status of the iAPX 86, iAPX 88, and the iAPX 186. Table 3 shows how the status lines are decoded. While in the Command mode the iAPX 286 status can be directly decoded. Microprocessor bus controller read or write commands or Multibus commands can also be directed to the 8207 when in Command mode.

Refresh Options

Immediately after system reset, the state of the REFRQ input pin is examined. If REFRQ is high, the 8207 provides the user with the choice between self-refresh or user-generated refresh with failsafe protection. Failsafe protection guarantees that if the

Table 3A. Status Coding of 8086, 80186 and 80286

Status Code			Function	
$\overline{S2}$	$\overline{S1}$	$\overline{S0}$	8086/80186	80286
0	0	0	INTERRUPT	INTERRUPT
0	0	1	I/O READ	I/O READ
0	1	0	I/O WRITE	I/O WRITE
0	1	1	HALT	IDLE
1	0	0	INSTRUCTION FETCH	HALT
1	0	1	MEMORY READ	MEMORY READ
1	1	0	MEMORY WRITE	MEMORY WRITE
1	1	1	IDLE	IDLE

Table 3B. 8207 Response

8207 Command			Function	
PCTL	$\overline{RD}$	$\overline{WR}$	8086/80186 Status Interface	80286/Status or Command Interface
0	0	0	IGNORE	IGNORE
0	0	1	IGNORE	READ
0	1	0	IGNORE	WRITE
0	1	1	IGNORE	IGNORE
1	0	0	READ	IGNORE
1	0	1	READ	INHIBIT
1	1	0	WRITE	INHIBIT
1	1	1	IGNORE	IGNORE

AFN-02218B

user does not come back with another refresh request before the internal refresh interval counter times out, a refresh request will be automatically generated. If the REFRQ pin is low immediately after a reset, then the user has the choice of a single external refresh cycle without failsafe, burst refresh or no refresh.

Internal Refresh Only

For the 8207 to generate internal refresh requests, it is necessary only to strap the REFRQ input pin high.

External Refresh with Failsafe

To allow user-generated refresh requests with failsafe protection, it is necessary to hold the REFRQ input high until after reset. Thereafter, a low-to-high transition on this input causes a refresh request to be generated and the internal refresh interval counter to be reset. A high-to-low transition has no effect on the 8207. A refresh request is not recognized until a previous request has been serviced.

External Refresh without Failsafe

To generate single external refresh requests without failsafe protection, it is necessary to hold REFRQ low until after reset. Thereafter, bringing REFRQ high for one clock period causes a refresh request to be generated. A refresh request is not recognized until a previous request has been serviced.

Burst Refresh

Burst refresh is implemented through the same procedure as a single external refresh without failsafe (i.e., REFRQ is kept low until after reset). Thereafter, bringing REFRQ high for a least two clock periods causes a burst of 128 row address locations to be refreshed.

In ECC-configured systems, 128 locations are scrubbed. A burst refresh request is not recognized until a previous request has been serviced.

No Refresh

It is necessary to hold REFRQ low until after reset. This is the same as programming External Refresh without Failsafe. No refresh is accomplished by keeping REFRQ low.

Option Program Data Word

The program data word consists of 16 program data bits, PD0–PD15. If the first program data bit PD0 is set to logic 1, the 8207 is configured to support ECC. If it is logic 0, the 8207 is configured to support a non-ECC system. The remaining bits, PD1–PD15, may then be programmed to optimize a selected configuration. Figures 9 and 10 show the Program word for non-ECC and ECC operation.

Using an External Shift Register

The 8207 may be configured to use an external shift register with asynchronous load capability such as a 74LS165. The reset pulse serves to parallel load the shift register and the 8207 supplies the clocking signal to shift the data in. Figure 11 shows a sample circuit diagram of an external shift register circuit. Serial data is shifted into the 8207 via the PDI pin (57), and clock is provided by the MUX/PCLK pin (12), which generates a total of 16 clock pulses. After programming is complete, data appearing at the input of the PDI pin is ignored. MUX/PCLK is a dual-function pin. During programming, it serves to clock the external shift register, and after programming is completed, it reverts to a MUX control pin. As the pin changes state to select different port addresses, it continues to clock the shift register. This does not present a problem because data at the PDI pin is ignored after programming. Figure 8 illustrates the timing requirements of the shift register circuitry.

ECC Mode (ECC Program Bit)

The state of PDI (Program Data In) pin at reset determines whether the system is an ECC or non-ECC configuration. It is used internally by the 8207 to begin configuring timing circuits, even before programming is completely finished. The 8207 then begins programming the rest of the options.

Default Programming Options

After reset, the 8207 serially shifts in a program data word via the PDI pin. This pin may be strapped either high or low, or connected to an external shift register. Strapping PDI high causes the 8207 to default to a particular system configuration with error correction, and strapping it low causes the 8207 to default to a particular system configuration without error correction. Table 4 shows the default configurations.

AFN-02218B

PD15							PD8	PD7							PD0
0	0	TM 1	$\overline{PPR}$	$\overline{FFS}$	EXT	$\overline{PLS}$	CI0	CI1	$\overline{RB1}$	$\overline{RB0}$	$\overline{RFS}$	$\overline{CFS}$	SB	$\overline{SA}$	0

PROGRAM DATA BIT	NAME	POLARITY/FUNCTION
PD0	ECC	ECC=0 FOR NON-ECC MODE
PD1	$\overline{SA}$	$\overline{SA}$=0 PORT A IS SYNCHRONOUS $\overline{SA}$=1 PORT A IS ASYNCHRONOUS
PD2	$\overline{SB}$	$\overline{SB}$=0 PORT B IS ASYNCHRONOUS $\overline{SB}$=1 PORT B IS SYNCHRONOUS
PD3	$\overline{CFS}$	$\overline{CFS}$=0 FAST-CYCLE iAPX 286 MODE $\overline{CFS}$=1 SLOW-CYCLE iAPX 86 MODE
PD4	$\overline{RFS}$	$\overline{RFS}$=0 FAST RAM $\overline{RFS}$=1 SLOW RAM
PD5 PD6	$\overline{RB0}$ $\overline{RB1}$	RAM BANK OCCUPANCY SEE TABLE 2
PD7	CI1	COUNT INTERVAL BIT 1; SEE TABLE 6
PD8	CI0	COUNT INTERVAL BIT 0; SEE TABLE 6
PD9	PLS	PLS=0 LONG REFRESH PERIOD PLS=1 SHORT REFRESH PERIOD
PD10	EXT $\overline{EXT}$	EXT=0 NOT EXTENDED EXT=1 EXTENDED
PD11	$\overline{FFS}$	$\overline{FFS}$=0 FAST CPU FREQUENCY $\overline{FFS}$=1 SLOW CPU FREQUENCY
PD12	$\overline{PPR}$	$\overline{PPR}$=0 MOST RECENTLY USED PORT PRIORITY $\overline{PPR}$=1 PORT A PREFERRED PRIORITY
PD13	TM1	TM1=0 TEST MODE 1 OFF TM1=1 TEST MODE 1 ENABLED
PD14	0	RESERVED MUST BE ZERO
PD15	0	RESERVED MUST BE ZERO

Figure 9. Non-ECC Mode Program Data Word

PD15							PD8	PD7							PD0
$\overline{TM2}$	RB1	RB0	PPR	FFS	$\overline{EXT}$	PLS	$\overline{CI0}$	$\overline{CI1}$	XB	$\overline{XA}$	RFS	CFS	$\overline{SB}$	SA	1

PROGRAM DATA BIT	NAME	POLARITY/FUNCTION
PD0	ECC	ECC=1 ECC MODE
PD1	SA	SA=0 PORT A ASYNCHRONOUS SA=1 PORT A SYNCHRONOUS
PD2	$\overline{SB}$	$\overline{SB}$=0 PORT B SYNCHRONOUS $\overline{SB}$=1 PORT B ASYNCHRONOUS
PD3	CFS	CFS=0 SLOW-CYCLE iAPX 86 MODE CFS=1 FAST-CYCLE iAPX 286 MODE
PD4	RFS	RFS=0 SLOW RAM RFS=1 FAST RAM
PD5	$\overline{XA}$	$\overline{XA}$=0 MULTIBUS-COMPATIBLE ACKA $\overline{XA}$=1 ADVANCED ACKA NOT MULTIBUS-COMPATIBLE
PD6	XB	XB=0 ADVANCED ACKB NOT MULTIBUS COMPATIBLE XB=1 MULTIBUS-COMPATIBLE ACKB
PD7	$\overline{CI1}$	COUNT INTERVAL BIT 1; SEE TABLE 6
PD8	$\overline{CI0}$	COUNT INTERVAL BIT 0; SEE TABLE 6
PD9	PLS	PLS=0 SHORT REFRESH PERIOD PLS=1 LONG REFRESH PERIOD
PD10	$\overline{EXT}$	$\overline{EXT}$=0 MASTER AND SLAVE EDCU $\overline{EXT}$=1 MASTER EDCU ONLY
PD11	FFS	FFS=0 SLOW CPU FREQUENCY FFS=1 FAST CPU FREQUENCY
PD12	PPR	PPR=0 PORT A PREFERRED PRIORITY PPR=1 MOST RECENTLY USED PORT PRIORITY
PD13 PD14	RB0 RB1	RAM BANK OCCUPANCY SEE TABLE 2
PD15	$\overline{TM2}$	$\overline{TM2}$=0 TEST MODE 2 ENABLED $\overline{TM2}$=1 TEST MODE 2 OFF

Figure 10. ECC Mode Program Data Word

AFN-02218B

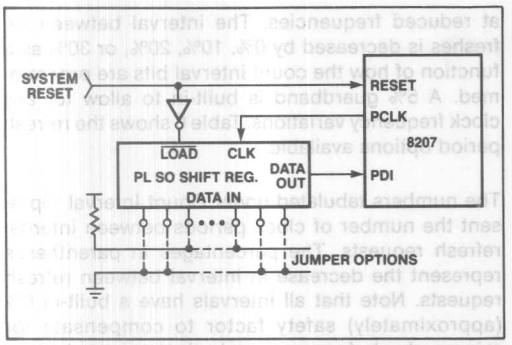


Figure 11. External Shift Register Interface

Table 4A.
Default Non-ECC Programming, PDI Pin (57) Tied to Ground.

Port A is Synchronous
Port B is Asynchronous
Fast-cycle Processor Interface
Fast RAM
Refresh Interval uses 236 clocks
128 Row refresh in 2 ms; 256 Row refresh in 4 ms
Fast Processor Clock Frequency (16 MHz)
"Most Recently Used" Priority Scheme
4 RAM banks occupied

Table 4B.
Default ECC Programming, PDI Pin (57) Tied to V_{CC}.

Port A is Synchronous
Port B is Asynchronous
Fast-cycle Processor Interface
Fast RAM
Port A has Advanced $\overline{ACKA}$ strobe (non-multibus)
Port B has Non-advance ACKB strobe (multibus)
Refresh interval uses 236 clocks
128 Row refresh in 2 ms; 256 Row refresh in 4 ms
Master EDCU only (16-bit system)
Fast Processor Clock Frequency (16 MHz)
"Most Recently Used" Priority Scheme
4 RAM banks occupied

If further system flexibility is needed, one or two external shift registers can be used to tailor the 8207 to its operating environment.

Synchronous/Asynchronous Mode (SA and SB Program Bits)

Each port of the 8207 may be independently configured to accept synchronous or asynchronous port commands ($\overline{RD}$, $\overline{WR}$, PCTL) and Port Enable ($\overline{PE}$) via the program bits SA and SB. The state of the SA and SB programming bits determine whether their associated ports are synchronous or asynchronous.

While a port may be configured with either the Status or Command interface in the synchronous mode, certain restrictions exist in the asynchronous mode. An asynchronous Command interface using the control lines of the Multibus is supported, and an asynchronous 8086 interface using the control lines of the 8086 is supported, with the use of TTL gates as illustrated in Figure 2. In the 8086 case, the TTL gates are needed to guarantee that status does not appear at the 8207 inputs too much before address, so that a cycle would start before address was valid.

Microprocessor Clock Frequency Option (CFS and FFS Program Bits)

The 8207 can be programmed to interface with slow-cycle microprocessors like the 8086, 8088, and 80186 or fast-cycle microprocessors like the 80286. The CFS bit configures the microprocessor interface to accept slow or fast cycle signals from either microprocessor group.

This option is used to select the speed of the microprocessor clock. Table 5 shows the various microprocessor clock frequency options that can be programmed.

Table 5.
Microprocessor Clock Frequency Options

Program Bits		Processor	Clock Frequency
CFS	FFS		
0	0	iAPX 86, 88, 186	5 MHz
0	1	iAPX 86, 88, 186	8 MHz
1	0	iAPX 286	10 MHz
1	1	iAPX 286	16 MHz

AFN-02218B

The external clock frequency must be programmed so that the failsafe refresh repetition circuitry can adjust its internal timing accordingly to produce a refresh request as programmed.

RAM Speed Option (RFS Program Bit)

The RAM Speed programming option determines whether RAM timing will be optimized for a fast or slow RAM. Whether a RAM is fast or slow is measured relative to the 2118-10 (Fast) or the 2118-15 (Slow) RAM specifications.

Refresh Period Options (CI0, CI1, and PLS Program Bits)

The 8207 refreshes with either 128 rows every 2 milliseconds or 256 rows every 4 milliseconds. This translates to one refresh cycle being executed approximately once every 15.6 microseconds. This rate can be changed to 256 rows every 2 milliseconds or a refresh approximately once every 7.8 microseconds via the Period Long/Short, program bit PLS, programming option. The 7.8 microsecond refresh request rate is intended for those RAMs, 64K and above, which may require a faster refresh rate.

In addition to PLS program option, two other programming bits for refresh exist: Count Interval 0 (CI0) and Count Interval 1 (CI1). These two programming bits allow the rate at which refresh requests are generated to be increased in order to permit refresh requests to be generated close to the same 15.6 or 7.8 microsecond period when the 8207 is operating at reduced frequencies. The interval between refreshes is decreased by 0%, 10%, 20%, or 30% as a function of how the count interval bits are programmed. A 5% guardband is built-in to allow for any clock frequency variations. Table 6 shows the refresh period options available.

The numbers tabulated under Count Interval represent the number of clock periods between internal refresh requests. The percentages in parentheses represent the decrease in interval between refresh requests. Note that all intervals have a built-in 5% (approximately) safety factor to compensate for minor clock frequency deviations and non-immediate response to internal refresh requests.

Extend Option (EXT Program Bit)

The Extend option lengthens the memory cycle to allow longer access time which may be required by the system. Extend alters the RAM timing to compensate for increased loading on the Row and Column Address Strobes, and in the multiplexed Address Out lines.

Port Priority Option and Arbitration (PPR Program Bit)

The 8207 has to internally arbitrate among three ports: Port A, Port B and Port C—the refresh port. Port C is an internal port dedicated to servicing refresh requests, whether they are generated internally by the refresh inverval counter, or externally by the user. Two arbitration approaches are available via

Table 6. Refresh Count Interval Table

Freq. (MHz)	Ref. Period (μS)	CFS	PLS	FFS	Count Interval CI1, CI0 (8207 Clock Periods)			
					00 (0%)	01 (10%)	10 (20%)	11 (30%)
16	15.6	1	1	1	236	212	188	164
	7.8	1	0	1	118	106	94	82
10	15.6	1	1	0	148	132	116	100
	7.8	1	0	0	74	66	58	50
8	15.6	0	1	1	118	106	94	82
	7.8	0	0	1	59	53	47	41
5	15.6	0	1	0	74	66	58	50
	7.8	0	0	0	37	33	29	25

AFN-02218B

the Port Priority programming option, program bit PPR. PPR determines whether the most recently used port will remain selected (PPR = 1) or whether Port A will be favored or preferred over Port B (PPR = 0).

A port is selected if the arbiter has given the selected port direct access to the timing generators. The front-end logic, which includes the arbiter, is designed to operate in parallel with the selected port. Thus a request on the selected port is serviced immediately. In contrast, an unselected port only has access to the timing generators through the front-end logic. Before a RAM cycle can start for an unselected port, that port must first become selected (i.e., the MUX output now gates that port's address into the 8207 in the case of Port A or B). Also, in order to allow its address to stabilize, a newly selected port's first RAM cycle is started by the front-end logic. Therefore, the selected port has direct access to the timing generators. What all this means is that a request on a selected port is started immediately, while a request on an unselected port is started two to three clock periods after the request, assuming that the other two ports are idle. Under normal operating conditions, this arbitration time is hidden behind the RAM cycle of the selected port so that as soon as the present cycle is over a new cycle is started. Table 7 lists the arbitration rules for both options.

Port LOCK Function

The LOCK function provides each port with the ability to obtain uninterrupted access to a critical region of memory and, thereby, to guarantee that the opposite port cannot "sneak in" and read from or write to the critical region prematurely.

Only one LOCK pin is present and is multiplexed between the two ports as follows: when MUX is high, the 8207 treats the LOCK input as originating at PORT A, while when MUX is low, the 8207 treats LOCK as originating at PORT B. When the 8207 recognizes a LOCK, the MUX output will remain pointed to the locking port until LOCK is deactivated. Refresh is not affected by LOCK and can occur during a locked memory cycle.

Table 7.
The Arbitration Rules for the Most Recently Used Port Priority and for Port A Priority Options Are As Follows:

1.	If only one port requests service, then that port—if not already selected—becomes selected.
2a.	When no service requests are pending, the last selected processor port (Port A or B) will remain selected. (Most Recently Used Port Priority Option)
2b.	When no service requests are pending, Port A is selected whether it requests service or not. (Port A Priority Option)
3.	During reset initialization only Port C, the refresh port, is selected.
4.	If no processor requests are pending after reset initialization, Port A will be selected.
5a.	If Ports A and B simultaneously(*) request service while Port C is being serviced, then the next port to be selected is the one which was not selected prior to servicing Port C. (Most Recently Used Port Priority Option)
5b.	If Ports A and B simultaneously(*) request service while Port C is selected, then the next port to be selected is Port A. (Port A Priority Option)
6.	If a port simultaneously requests service with the currently selected port, service is granted to the selected port.
7.	The MUX output remains in its last state whenever Port C is selected.
8.	If Port C and either Port A or Port B (or both) simultaneously request service, then service is granted to the requester whose port is already selected. If the selected port is not requesting service, then service is granted to Port C.
9.	If during the servicing of one port, the other port requests service before or simultaneously with the refresh port, the refresh port is selected. A new port is not selected before the presently selected port is deactivated.
10.	Activating LOCK will mask off service requests from Port B if the MUX output is high, or from Port A if the MUX output is low.
* By "simultaneous" it is meant that two or more requests are valid at the clock edge at which the internal arbiter samples them.	

AFN-02218B

the Port Priority programming option, program bit PPR. PPR determines whether the most recently used port will remain selected (PPR = 1) or whether Port A will be favored or preferred over Port B (PPR = 0).

A port is selected if the arbiter has given the selected port direct access to the timing generators. The front-end logic, which includes the arbiter, is designed to operate in parallel with the selected port. Thus a request on the selected port is serviced immediately. In contrast, an unselected port only has access to the timing generators through the front-end logic. Before a RAM cycle can start for an unselected port, that port must first become selected (i.e., the MUX output now gates that port's address into the 8207. In the case of Port A or B). Also, in order to allow its address to stabilize, a newly selected port's first RAM cycle is started by the front-end logic. Therefore, the selected port has direct access to the timing generators. What all this means is that a request on a selected port is started immediately while a request on an unselected port is started two to three clock periods after the request, assuming that the other two ports are idle. Under normal operating conditions, this arbitration time is hidden behind the RAM cycle of the selected port so that as soon as the present cycle is over a new cycle is started. Table 7 lists the arbitration rules for both options.

Port LOCK Function

The LOCK function provides each port with the ability to obtain uninterrupted access to a critical region of memory and, thereby, to guarantee that the opposite port cannot "sneak in" and read from or write to the critical region prematurely.

Only one LOCK pin is present and is multiplexed between the two ports as follows: when MUX is high, the 8207 treats the LOCK input as originating at PORT A, while when MUX is low, the 8207 treats LOCK as originating at PORT B. When the 8207 recognizes a LOCK, the MUX output will remain pointed to the locking port until LOCK is deactivated. Refresh is not affected by LOCK and can occur during a locked memory cycle.

Table 7.
The Arbitration Rules for the Most Recently Used Port Priority and for Port A Priority Options Are As Follows:

	If only one port requests service, then that port—if not already selected—becomes selected.
2a.	When no service requests are pending, the last selected processor port (Port A or B) will remain selected. (Most Recently Used Port Priority Option)
2b.	When no service requests are pending, Port A is selected whether it requests service or not. (Port A Priority Option)
3.	During reset initialization only Port C, the refresh port, is selected.
4.	If no processor requests are pending after reset initialization, Port A will be selected.
5a.	If Ports A and B simultaneously(*) request service while Port C is being serviced, then the next port to be selected is the one which was not selected prior to servicing Port C. (Most Recently Used Port Priority Option)
5b.	If Ports A and B simultaneously(*) request service while Port C is selected, then the next port to be selected is Port A. (Port A Priority Option)
6.	If a port simultaneously requests service with the currently selected port, service is granted to the selected port.
7.	The MUX output remains in its last state whenever Port C is selected.
8.	If Port C and either Port A or Port B (or both) simultaneously request service, then service is granted to the requester whose port is already selected. If the selected port is not requesting service, then service is granted to Port C.
9.	If during the servicing of one port, the other port requests service before or simultaneously with the refresh port, the refresh port is selected. A new port is not selected before the presently selected port is deactivated.
10.	Activating LOCK will mask off service requests from Port B if the MUX output is high, or from Port A if the MUX output is low.

* By "simultaneously" is meant that two or more requests are valid at the clock edge at which the internal arbiter samples them.

A27-02218

8259A/8259A-2/8259A-8
PROGRAMMABLE INTERRUPT CONTROLLER

- iAPX 86, iAPX 88 Compatible
- MCS-80®, MCS-85® Compatible
- Eight-Level Priority Controller
- Expandable to 64 Levels
- Programmable Interrupt Modes
- Individual Request Mask Capability
- Single +5V Supply (No Clocks)
- 28-Pin Dual-In-Line Package
- Available in EXPRESS
 - Standard Temperature Range
 - Extended Temperature Range

The Intel® 8259A Programmable Interrupt Controller handles up to eight vectored priority interrupts for the CPU. It is cascadable for up to 64 vectored priority interrupts without additional circuitry. It is packaged in a 28-pin DIP, uses NMOS technology and requires a single +5V supply. Circuitry is static, requiring no clock input.

The 8259A is designed to minimize the software and real time overhead in handling multi-level priority interrupts. It has several modes, permitting optimization for a variety of system requirements.

The 8259A is fully upward compatible with the Intel® 8259. Software originally written for the 8259 will operate the 8259A in all 8259 equivalent modes (MCS-80/85, Non-Buffered, Edge Triggered).

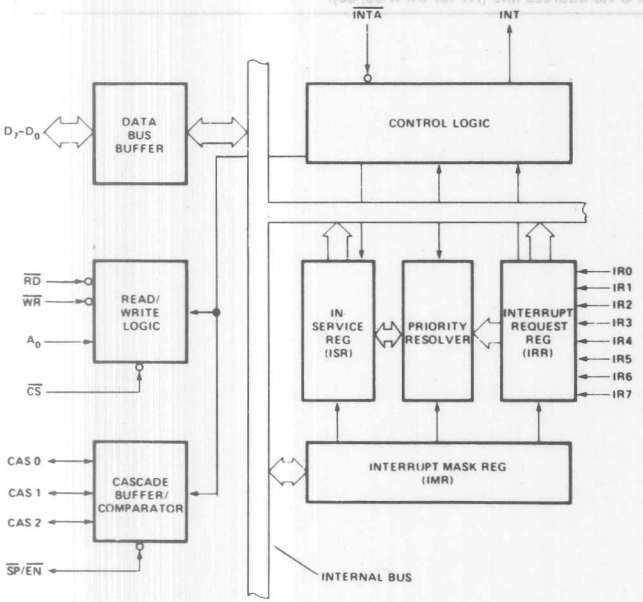


Figure 1. Block Diagram

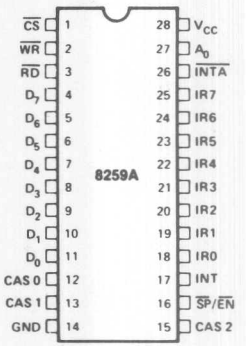
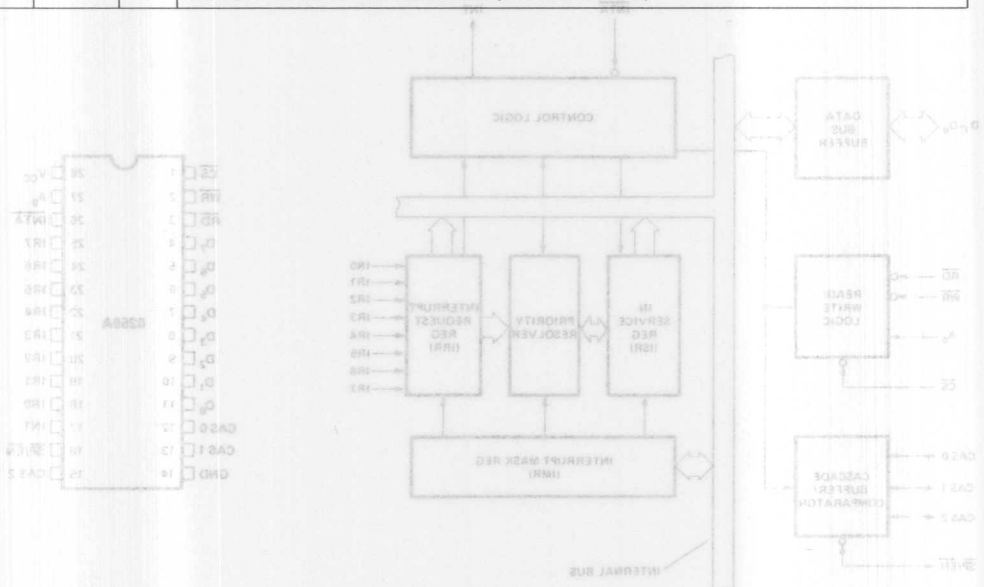


Figure 2. Pin Configuration

Intel Corporation Assumes No Responsibilty for the Use of Any Circuitry Other Than Circuitry Embodied in an Intel Product. No Other Circuit Patent Licenses are Implied.
© INTEL CORPORATION, 1980
AFN-00221C

Table 1. Pin Description

Symbol	Pin No.	Type	Name and Function
V_{CC}	28	I	**Supply:** +5V Supply.
GND	14	I	**Ground.**
$\overline{CS}$	1	I	**Chip Select:** A low on this pin enables $\overline{RD}$ and $\overline{WR}$ communication between the CPU and the 8259A. INTA functions are independent of CS.
$\overline{WR}$	2	O	**Write:** A low on this pin when CS is low enables the 8259A to accept command words from the CPU.
$\overline{RD}$	3	I	**Read:** A low on this pin when CS is low enables the 8259A to release status onto the data bus for the CPU.
D_7–D_0	4–11	I/O	**Bidirectional Data Bus:** Control, status and interrupt-vector information is transferred via this bus.
CAS_0–CAS_2	12, 13, 15	I/O	**Cascade Lines:** The CAS lines form a private 8259A bus to control a multiple 8259A structure. These pins are outputs for a master 8259A and inputs for a slave 8259A.
$\overline{SP}/\overline{EN}$	16	I/O	**Slave Program/Enable Buffer:** This is a dual function pin. When in the Buffered Mode it can be used as an output to control buffer transceivers (EN). When not in the buffered mode it is used as an input to designate a master (SP = 1) or slave (SP = 0).
INT	17	O	**Interrupt:** This pin goes high whenever a valid interrupt request is asserted. It is used to interrupt the CPU, thus it is connected to the CPU's interrupt pin.
IR_0–IR_7	18–25	I	**Interrupt Requests:** Asynchronous inputs. An interrupt request is executed by raising an IR input (low to high), and holding it high until it is acknowledged (Edge Triggered Mode), or just by a high level on an IR input (Level Triggered Mode).
$\overline{INTA}$	26	I	**Interrupt Acknowledge:** This pin is used to enable 8259A interrupt-vector data onto the data bus by a sequence of interrupt acknowledge pulses issued by the CPU.
A_0	27	I	**AO Address Line:** This pin acts in conjunction with the $\overline{CS}$, $\overline{WR}$, and $\overline{RD}$ pins. It is used by the 8259A to decipher various Command Words the CPU writes and status the CPU wishes to read. It is typically connected to the CPU A0 address line (A1 for iAPX 86, 88).



AFN-00221E

FUNCTIONAL DESCRIPTION

Interrupts in Microcomputer Systems

Microcomputer system design requires that I/O devices such as keyboards, displays, sensors and other components receive servicing in an efficient manner so that large amounts of the total system tasks can be assumed by the microcomputer with little or no effect on throughput.

The most common method of servicing such devices is the *Polled* approach. This is where the processor must test each device in sequence and in effect "ask" each one if it needs servicing. It is easy to see that a large portion of the main program is looping through this continuous polling cycle and that such a method would have a serious, detrimental effect on system throughput, thus limiting the tasks that could be assumed by the microcomputer and reducing the cost effectiveness of using such devices.

A more desirable method would be one that would allow the microprocessor to be executing its main program and only stop to service peripheral devices when it is told to do so by the device itself. In effect, the method would provide an external asynchronous input that would inform the processor that it should complete whatever instruction that is currently being executed and fetch a new routine that will service the requesting device. Once this servicing is complete, however, the processor would resume exactly where it left off.

This method is called *Interrupt*. It is easy to see that system throughput would drastically increase, and thus more tasks could be assumed by the microcomputer to further enhance its cost effectiveness.

The Programmable Interrupt Controller (PIC) functions as an overall manager in an Interrupt-Driven system environment. It accepts requests from the peripheral equipment, determines which of the incoming requests is of the highest importance (priority), ascertains whether the incoming request has a higher priority value than the level currently being serviced, and issues an interrupt to the CPU based on this determination.

Each peripheral device or structure usually has a special program or "routine" that is associated with its specific functional or operational requirements; this is referred to as a "service routine". The PIC, after issuing an Interrupt to the CPU, must somehow input information into the CPU that can "point" the Program Counter to the service routine associated with the requesting device. This "pointer" is an address in a vectoring table and will often be referred to, in this document, as vectoring data.

The 8259A

The 8259A is a device specifically designed for use in real time, interrupt driven microcomputer systems. It manages eight levels or requests and has built-in features for expandability to other 8259A's (up to 64 levels). It is programmed by the system's software as an I/O peripheral. A selection of priority modes is available to the programmer so that the manner in which the requests are processed by the 8259A can be configured to match his system requirements. The priority modes can be changed or reconfigured dynamically at any time during the main program. This means that the complete interrupt structure can be defined as required, based on the total system environment.

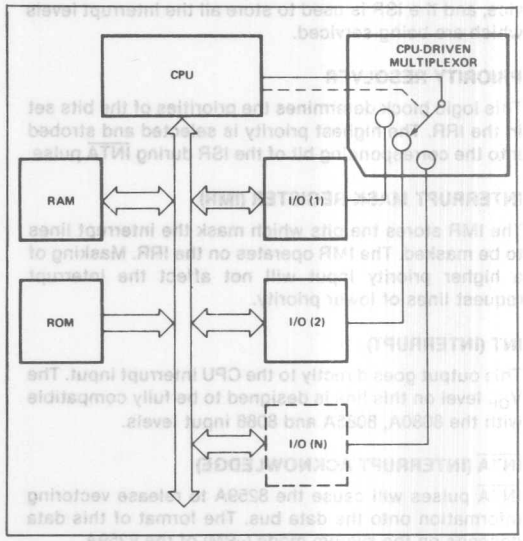


Figure 3a. Polled Method

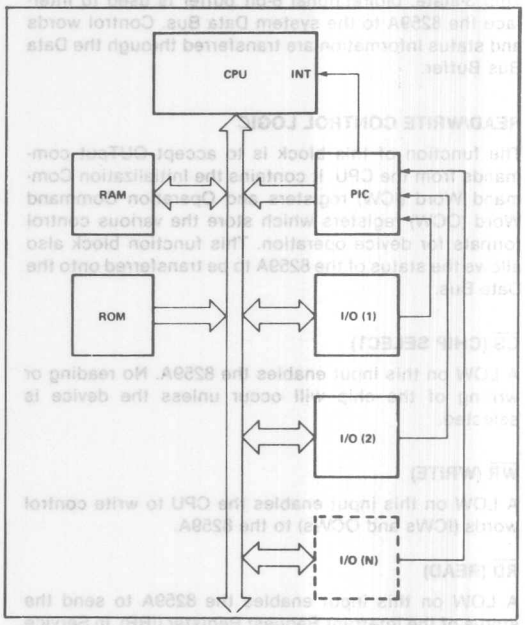


Figure 3b. Interrupt Method

AFN-00221E

IN-SERVICE REGISTER (ISR)

The interrupts at the IR input lines are handled by two registers in cascade, the Interrupt Request Register (IRR) and the In-Service Register (ISR). The IRR is used to store all the interrupt levels which are requesting service; and the ISR is used to store all the interrupt levels which are being serviced.

PRIORITY RESOLVER

This logic block determines the priorities of the bits set in the IRR. The highest priority is selected and strobed into the corresponding bit of the ISR during $\overline{INTA}$ pulse.

INTERRUPT MASK REGISTER (IMR)

The IMR stores the bits which mask the interrupt lines to be masked. The IMR operates on the IRR. Masking of a higher priority input will not affect the interrupt request lines of lower priority.

INT (INTERRUPT)

This output goes directly to the CPU interrupt input. The V_{OH} level on this line is designed to be fully compatible with the 8080A, 8085A and 8086 input levels.

$\overline{INTA}$ (INTERRUPT ACKNOWLEDGE)

$\overline{INTA}$ pulses will cause the 8259A to release vectoring information onto the data bus. The format of this data depends on the system mode (μPM) of the 8259A.

DATA BUS BUFFER

This 3-state, bidirectional 8-bit buffer is used to interface the 8259A to the system Data Bus. Control words and status information are transferred through the Data Bus Buffer.

READ/WRITE CONTROL LOGIC

The function of this block is to accept OUTput commands from the CPU. It contains the Initialization Command Word (ICW) registers and Operation Command Word (OCW) registers which store the various control formats for device operation. This function block also allows the status of the 8259A to be transferred onto the Data Bus.

$\overline{CS}$ (CHIP SELECT)

A LOW on this input enables the 8259A. No reading or writing of the chip will occur unless the device is selected.

$\overline{WR}$ (WRITE)

A LOW on this input enables the CPU to write control words (ICWs and OCWs) to the 8259A.

$\overline{RD}$ (READ)

A LOW on this input enables the 8259A to send the status of the Interrupt Request Register (IRR), In Service Register (ISR), the Interrupt Mask Register (IMR), or the Interrupt level onto the Data Bus.

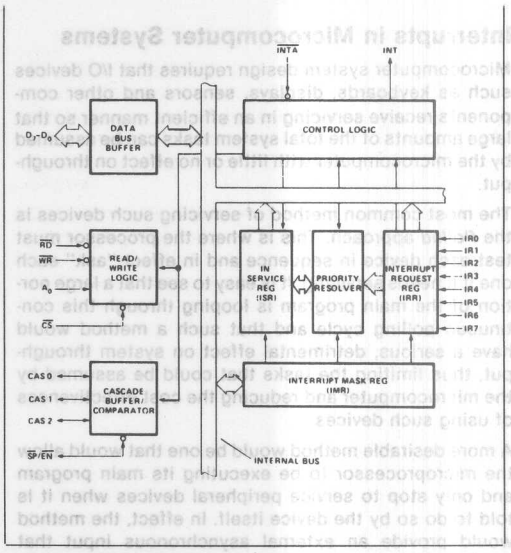


Figure 4a. 8259A Block Diagram

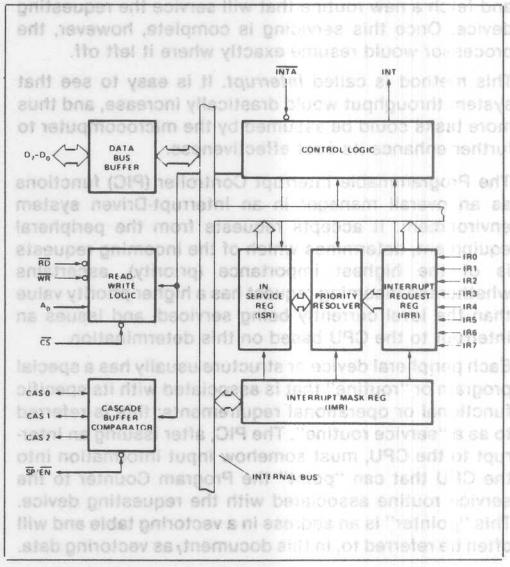


Figure 4b. 8259A Block Diagram

A_0

This input signal is used in conjunction with $\overline{WR}$ and $\overline{RD}$ signals to write commands into the various command registers, as well as reading the various status registers of the chip. This line can be tied directly to one of the address lines.

AFN-00221E

THE CASCADE BUFFER/COMPARATOR

This function block stores and compares the IDs of all 8259A's used in the system. The associated three I/O pins (CAS0-2) are outputs when the 8259A is used as a master and are inputs when the 8259A is used as a slave. As a master, the 8259A sends the ID of the interrupting slave device onto the CAS0-2 lines. The slave thus selected will send its preprogrammed subroutine address onto the Data Bus during the next one or two consecutive $\overline{INTA}$ pulses. (See section "Cascading the 8259A".)

INTERRUPT SEQUENCE

The powerful features of the 8259A in a microcomputer system are its programmability and the interrupt routine addressing capability. The latter allows direct or indirect jumping to the specific interrupt routine requested without any polling of the interrupting devices. The normal sequence of events during an interrupt depends on the type of CPU being used.

The events occur as follows in an MCS-80/85 system:

1. One or more of the INTERRUPT REQUEST lines (IR7-0) are raised high, setting the corresponding IRR bit(s).
2. The 8259A evaluates these requests, and sends an INT to the CPU, if appropriate
3. The CPU acknowledges the INT and responds with an $\overline{INTA}$ pulse.
4. Upon receiving an $\overline{INTA}$ from the CPU group, the highest priority ISR bit is set, and the corresponding IRR bit is reset. The 8259A will also release a CALL instruction code (11001101) onto the 8-bit Data Bus through its D7-0 pins.
5. This CALL instruction will initiate two more $\overline{INTA}$ pulses to be sent to the 8259A from the CPU group.
6. These two $\overline{INTA}$ pulses allow the 8259A to release its preprogrammed subroutine address onto the Data Bus. The lower 8-bit address is released at the first $\overline{INTA}$ pulse and and the higher 8-bit address is released at the second $\overline{INTA}$ pulse.
7. This completes the 3-byte CALL instruction released by the 8259A. In the AEOI mode the ISR bit is reset at the end of the third $\overline{INTA}$ pulse. Otherwise, the ISR bit remains set until an appropriate EOI command is issued at the end of the interrupt sequence.

The events occurring in an iAPX 86 system are the same until step 4.

4. Upon receiving an $\overline{INTA}$ from the CPU group, the highest priority ISR bit is set and the corresponding IRR bit is reset. The 8259A does not drive the Data Bus during this cycle.
5. The iAPX 86/10 will initiate a second $\overline{INTA}$ pulse. During this pulse, the 8259A releases an 8-bit pointer onto the Data Bus where it is read by the CPU
6. This completes the interrupt cycle. In the AEOI mode the ISR bit is reset at the end of the second $\overline{INTA}$ pulse. Otherwise, the ISR bit remains set until an appropriate EOI command is issued at the end of the interrupt subroutine.

If no interrupt request is present at step 4 of either sequence (i.e., the request was too short in duration) the 8259A will issue an interrupt level 7. Both the vectoring bytes and the CAS lines will look like an interrupt level 7 was requested.

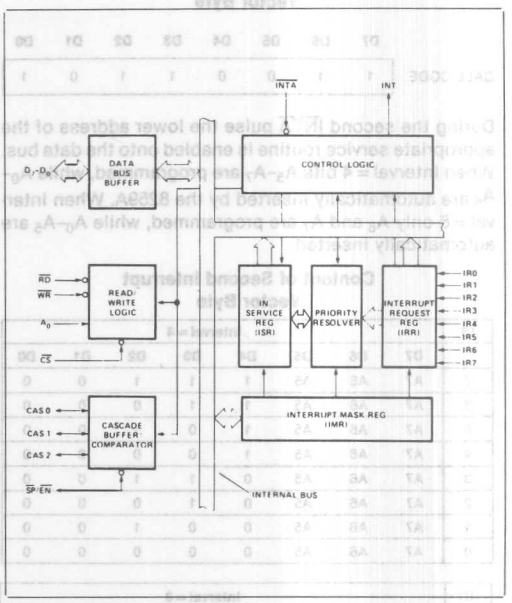


Figure 4c. 8259A Block Diagram

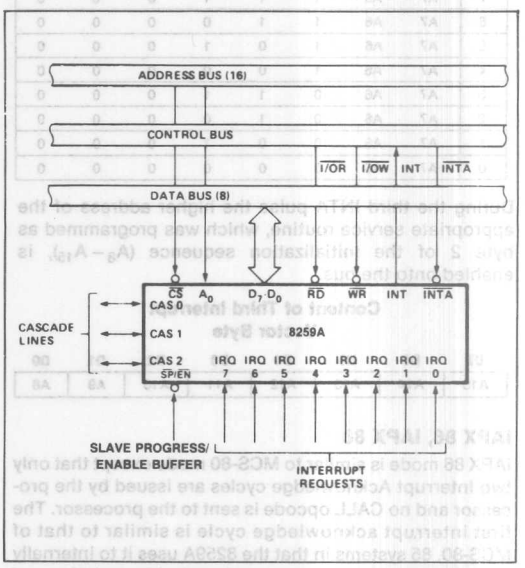


Figure 5. 8259A Interface to Standard System Bus

AFN 00221E

INTERRUPT SEQUENCE OUTPUTS

MCS-80®, MCS-85®

This sequence is timed by three $\overline{INTA}$ pulses. During the first $\overline{INTA}$ pulse the CALL opcode is enabled onto the data bus.

Content of First Interrupt Vector Byte

	D7	D6	D5	D4	D3	D2	D1	D0
CALL CODE	1	1	0	0	1	1	0	1

During the second $\overline{INTA}$ pulse the lower address of the appropriate service routine is enabled onto the data bus. When Interval = 4 bits A_5–A_7 are programmed, while A_0–A_4 are automatically inserted by the 8259A. When Interval = 8 only A_6 and A_7 are programmed, while A_0–A_5 are automatically inserted.

Content of Second Interrupt Vector Byte

IR	Interval = 4							
	D7	D6	D5	D4	D3	D2	D1	D0
7	A7	A6	A5	1	1	1	0	0
6	A7	A6	A5	1	1	0	0	0
5	A7	A6	A5	1	0	1	0	0
4	A7	A6	A5	1	0	0	0	0
3	A7	A6	A5	0	1	1	0	0
2	A7	A6	A5	0	1	0	0	0
1	A7	A6	A5	0	0	1	0	0
0	A7	A6	A5	0	0	0	0	0

IR	Interval = 8							
	D7	D6	D5	D4	D3	D2	D1	D0
7	A7	A6	1	1	1	0	0	0
6	A7	A6	1	1	0	0	0	0
5	A7	A6	1	0	1	0	0	0
4	A7	A6	1	0	0	0	0	0
3	A7	A6	0	1	1	0	0	0
2	A7	A6	0	1	0	0	0	0
1	A7	A6	0	0	1	0	0	0
0	A7	A6	0	0	0	0	0	0

During the third INTA pulse the higher address of the appropriate service routine, which was programmed as byte 2 of the initialization sequence (A_8 – A_{15}), is enabled onto the bus.

Content of Third Interrupt Vector Byte

D7	D6	D5	D4	D3	D2	D1	D0
A15	A14	A13	A12	A11	A10	A9	A8

iAPX 86, iAPX 88

iAPX 86 mode is similar to MCS-80 mode except that only two Interrupt Acknowledge cycles are issued by the processor and no CALL opcode is sent to the processor. The first interrupt acknowledge cycle is similar to that of MCS-80, 85 systems in that the 8259A uses it to internally freeze the state of the interrupts for priority resolution and as a master it issues the interrupt code on the cascade lines at the end of the INTA pulse. On this first cycle it does not issue any data to the processor and leaves its data bus buffers disabled. On the second interrupt acknowledge cycle in iAPX 86 mode the master (or slave if so programmed) will send a byte of data to the processor with the acknowledged interrupt code composed as follows (note the state of the ADI mode control is ignored and A_5–A_{11} are unused in iAPX 86 mode):

Content of Interrupt Vector Byte for iAPX 86 System Mode

	D7	D6	D5	D4	D3	D2	D1	D0
IR7	T7	T6	T5	T4	T3	1	1	1
IR6	T7	T6	T5	T4	T3	1	1	0
IR5	T7	T6	T5	T4	T3	1	0	1
IR4	T7	T6	T5	T4	T3	1	0	0
IR3	T7	T6	T5	T4	T3	0	1	1
IR2	T7	T6	T5	T4	T3	0	1	0
IR1	T7	T6	T5	T4	T3	0	0	1
IR0	T7	T6	T5	T4	T3	0	0	0

PROGRAMMING THE 8259A

The 8259A accepts two types of command words generated by the CPU:

1. *Initialization Command Words (ICWs):* Before normal operation can begin, each 8259A in the system must be brought to a starting point — by a sequence of 2 to 4 bytes timed by $\overline{WR}$ pulses.
2. *Operation Command Words (OCWs):* These are the command words which command the 8259A to operate in various interrupt modes. These modes are:
 a. Fully nested mode
 b. Rotating priority mode
 c. Special mask mode
 d. Polled mode

The OCWs can be written into the 8259A anytime after initialization.

INITIALIZATION COMMAND WORDS (ICWS)

GENERAL

Whenever a command is issued with A0 = 0 and D4 = 1, this is interpreted as Initialization Command Word 1 (ICW1). ICW1 starts the initialization sequence during which the following automatically occur.

a. The edge sense circuit is reset, which means that following initialization, an interrupt request (IR) input must make a low-to-high transition to generate an interrupt.
b. The Interrupt Mask Register is cleared.
c. IR7 input is assigned priority 7.
d. The slave mode address is set to 7.
e. Special Mask Mode is cleared and Status Read is set to IRR.
f. If IC4=0, then all functions selected in ICW4 are set to zero. (Non-Buffered mode*, no Auto-EOI, MCS-80, 85 system).

*Note: Master/Slave in ICW4 is only used in the buffered mode.

AFN-00221E

INITIALIZATION COMMAND WORDS 1 AND 2 (ICW1, ICW2)

A_5-A_{15}: *Page starting address of service routines.* In an MCS 80/85 system, the 8 request levels will generate CALLs to 8 locations equally spaced in memory. These can be programmed to be spaced at intervals of 4 or 8 memory locations, thus the 8 routines will occupy a page of 32 or 64 bytes, respectively.

The address format is 2 bytes long (A_0-A_{15}). When the routine interval is 4, A_0-A_4 are automatically inserted by the 8259A, while A_5-A_{15} are programmed externally. When the routine interval is 8, A_0-A_5 are automatically inserted by the 8259A, while A_6-A_{15} are programmed externally.

The 8-byte interval will maintain compatibility with current software, while the 4-byte interval is best for a compact jump table.

In an iAPX 86 system A_{15}-A_{11} are inserted in the five most significant bits of the vectoring byte and the 8259A sets the three least significant bits according to the interrupt level. A_{10}-A_5 are ignored and ADI (Address interval) has no effect.

LTIM: If LTIM = 1, then the 8259A will operate in the level interrupt mode. Edge detect logic on the interrupt inputs will be disabled.

ADI: CALL address interval. ADI = 1 then interval = 4; ADI = 0 then interval = 8.

SNGL: Single. Means that this is the only 8259A in the system. If SNGL = 1 no ICW3 will be issued.

IC4: If this bit is set — ICW4 has to be read. If ICW4 is not needed, set IC4 = 0.

INITIALIZATION COMMAND WORD 3 (ICW3)

This word is read only when there is more than one 8259A in the system and cascading is used, in which case SNGL = 0. It will load the 8-bit slave register. The functions of this register are:

a. In the master mode (either when SP = 1, or in buffered mode when M/S = 1 in ICW4) a "1" is set for each slave in the system. The master then will release byte 1 of the call sequence (for MCS-80/85 system) and will enable the corresponding slave to release bytes 2 and 3 (for iAPX 86 only byte 2) through the cascade lines.

b. In the slave mode (either when $\overline{SP}$ = 0, or if BUF = 1 and M/S = 0 in ICW4) bits 2-0 identify the slave. The slave compares its cascade input with these bits and, if they are equal, bytes 2 and 3 of the call sequence (or just byte 2 for iAPX 86 are released by it on the Data Bus.

INITIALIZATION COMMAND WORD 4 (ICW4)

SFNM: If SFNM = 1 the special fully nested mode is programmed.

BUF: If BUF = 1 the buffered mode is programmed. In buffered mode $\overline{SP}/\overline{EN}$ becomes an enable output and the master/slave determination is by M/S.

M/S: If buffered mode is selected: M/S = 1 means the 8259A is programmed to be a master, M/S = 0 means the 8259A is programmed to be a slave. If BUF = 0, M/S has no function.

AEOI: If AEOI = 1 the automatic end of interrupt mode is programmed.

μPM: Microprocessor mode: μPM = 0 sets the 8259A for MCS-80, 85 system operation, μPM = 1 sets the 8259A for iAPX 86 system operation.

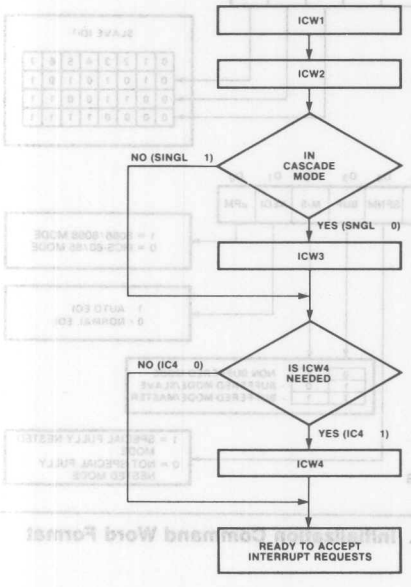


Figure 6. Initialization Sequence

AFN-00221E

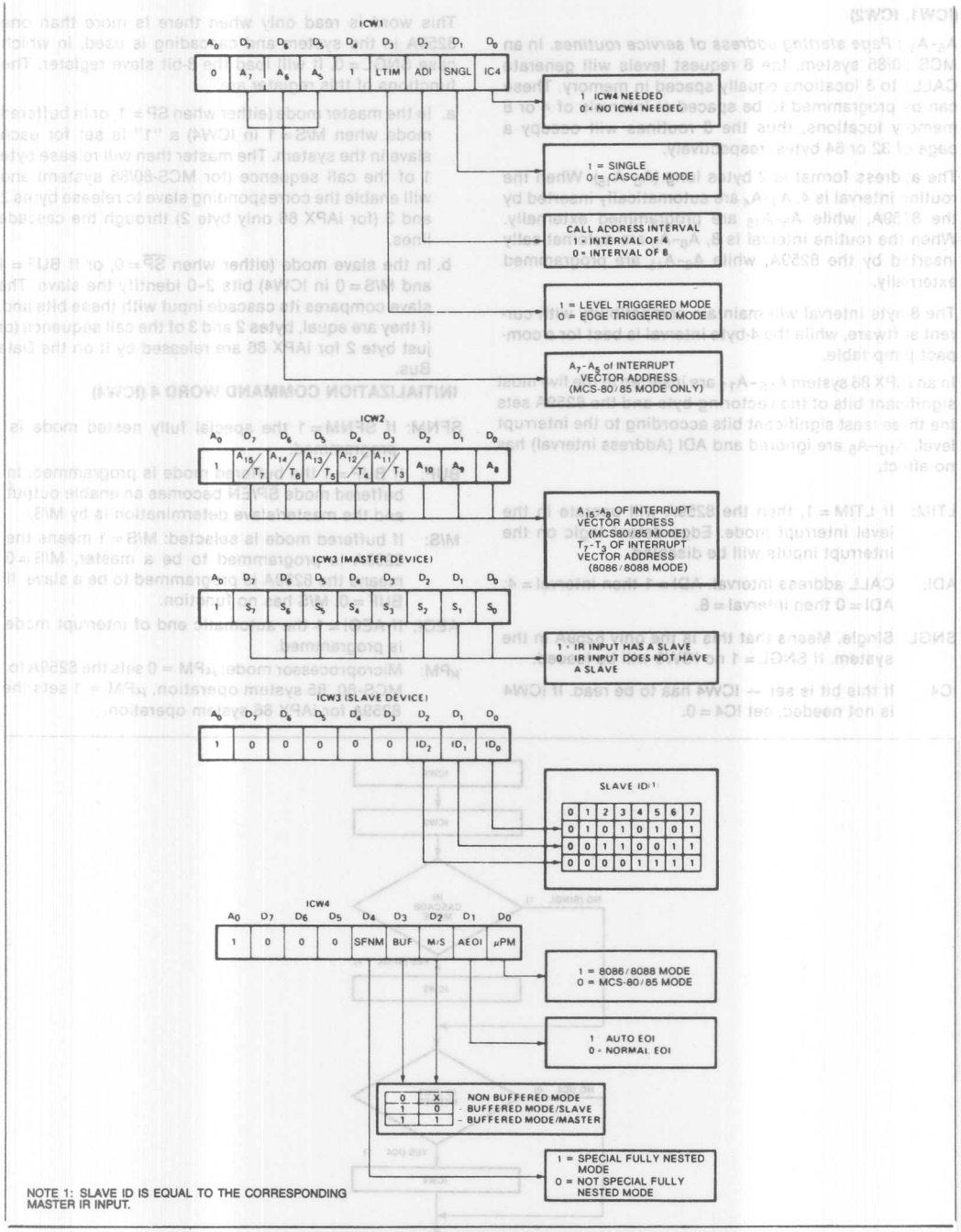


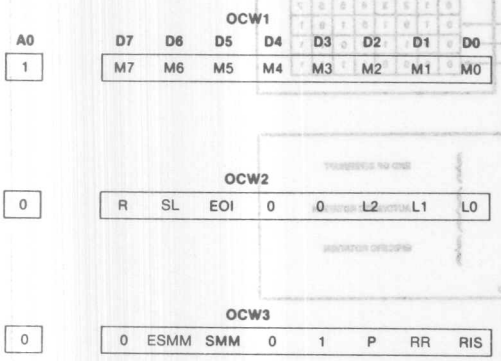
Figure 7. Initialization Command Word Format

AFN-00221E

OPERATION COMMAND WORDS (OCWs)

After the Initialization Command Words (ICWs) are programmed into the 8259A, the chip is ready to accept interrupt requests at its input lines. However, during the 8259A operation, a selection of algorithms can command the 8259A to operate in various modes through the Operation Command Words (OCWs).

OPERATION CONTROL WORDS (OCWs)



OPERATION CONTROL WORD 1 (OCW1)

OCW1 sets and clears the mask bits in the interrupt Mask Register (IMR). $M_7 - M_0$ represent the eight mask bits. M = 1 indicates the channel is masked (inhibited), M = 0 indicates the channel is enabled.

OPERATION CONTROL WORD 2 (OCW2)

R, SL, EOI — These three bits control the Rotate and End of Interrupt modes and combinations of the two. A chart of these combinations can be found on the Operation Command Word Format.

L_2, L_1, L_0—These bits determine the interrupt level acted upon when the SL bit is active.

OPERATION CONTROL WORD 3 (OCW3)

ESMM — Enable Special Mask Mode. When this bit is set to 1 it enables the SMM bit to set or reset the Special Mask Mode. When ESMM = 0 the SMM bit becomes a "don't care".

SMM — Special Mask Mode. If ESMM = 1 and SMM = 1 the 8259A will enter Special Mask Mode. If ESMM = 1 and SMM = 0 the 8259A will revert to normal mask mode. When ESMM = 0, SMM has no effect.

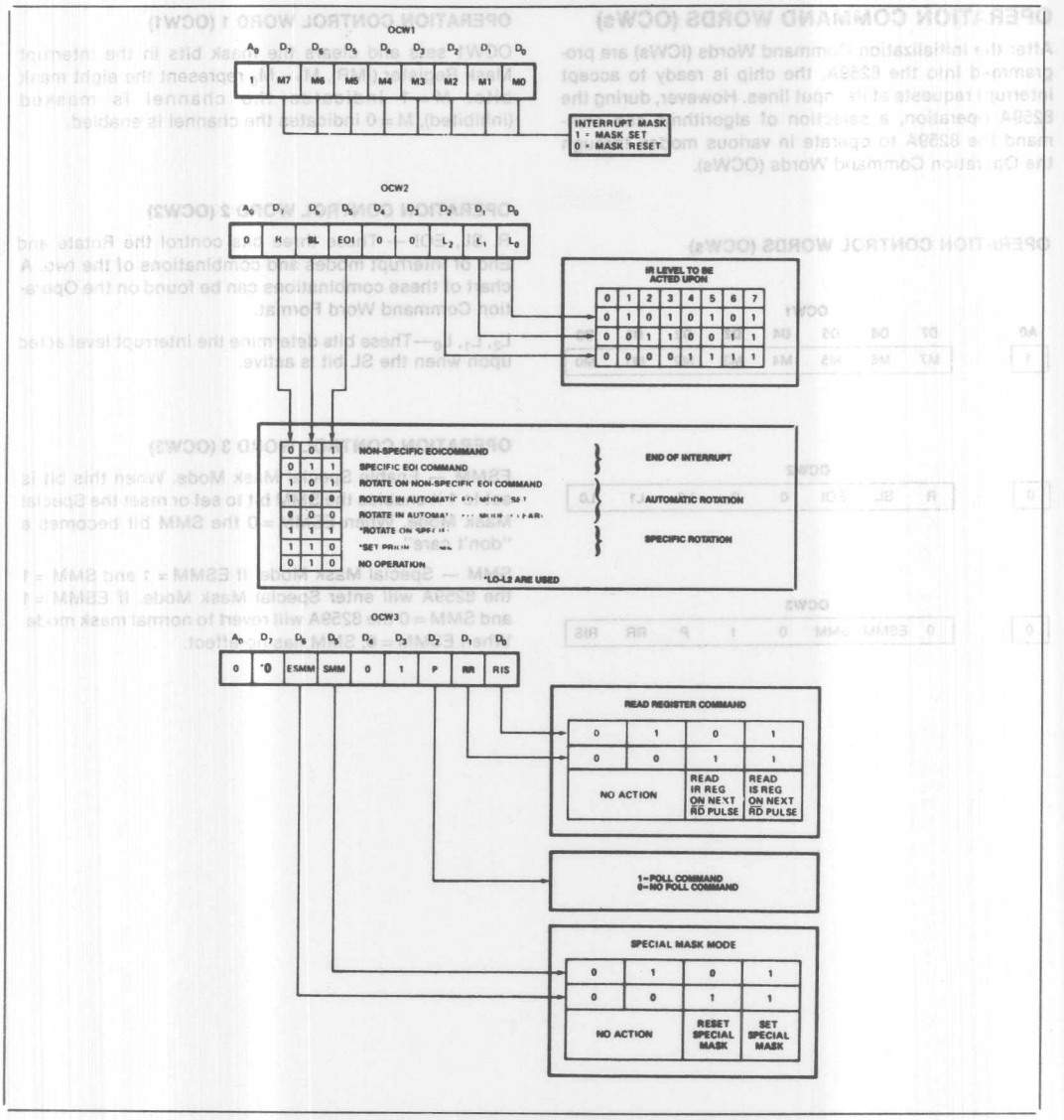


Figure 8. Operation Command Word Format

AFN-00221E

FULLY NESTED MODE

This mode is entered after initialization unless another mode is programmed. The interrupt requests are ordered in priority form 0 through 7 (0 highest). When an interrupt is acknowledged the highest priority request is determined and its vector placed on the bus. Additionally, a bit of the Interrupt Service register (ISO-7) is set. This bit remains set until the microprocessor issues an End of Interrupt (EOI) command immediately before returning from the service routine, or if AEOI (Automatic End of Interrupt) bit is set, until the trailing edge of the last INTA. While the IS bit is set, all further interrupts of the same or lower priority are inhibited, while higher levels will generate an interrupt (which will be acknowledged only if the microprocessor internal Interrupt enable flip-flop has been re-enabled through software).

After the initialization sequence, IR0 has the highest priority and IR7 the lowest. Priorities can be changed, as will be explained, in the rotating priority mode.

END OF INTERRUPT (EOI)

The In Service (IS) bit can be reset either automatically following the trailing edge of the last in sequence INTA pulse (when AEOI bit in ICW1 is set) or by a command word that must be issued to the 8259A before returning from a service routine (EOI command). An EOI command must be issued twice if in the Cascade mode, once for the master and once for the corresponding slave.

There are two forms of EOI command: Specific and Non-Specific. When the 8259A is operated in modes which preserve the fully nested structure, it can determine which IS bit to reset on EOI. When a Non-Specific EOI command is issued the 8259A will automatically reset the highest IS bit of those that are set, since in the fully nested mode the highest IS level was necessarily the last level acknowledged and serviced. A non-specific EOI can be issued with OCW2 (EOI = 1, SL = 0, R = 0).

When a mode is used which may disturb the fully nested structure, the 8259A may no longer be able to determine the last level acknowledged. In this case a Specific End of Interrupt must be issued which includes as part of the command the IS level to be reset. A specific EOI can be issued with OCW2 (EOI = 1, SL = 1, R = 0, and LO-L2 is the binary level of the IS bit to be reset).

It should be noted that an IS bit that is masked by an IMR bit will not be cleared by a non-specific EOI if the 8259A is in the Special Mask Mode.

AUTOMATIC END OF INTERRUPT (AEOI) MODE

If AEOI = 1 in ICW4, then the 8259A will operate in AEOI mode continuously until reprogrammed by ICW4. In this mode the 8259A will automatically perform a non-specific EOI operation at the trailing edge of the last interrupt acknowledge pulse (third pulse in MCS-80/85, second in iAPX 86). Note that from a system standpoint, this mode should be used only when a nested multilevel interrupt structure is not required within a single 8259A.

The AEOI mode can only be used in a master 8259A and not a slave.

AUTOMATIC ROTATION (Equal Priority Devices)

In some applications there are a number of interrupting devices of equal priority. In this mode a device, after being serviced, receives the lowest priority, so a device requesting an interrupt will have to wait, in the worst case until each of 7 other devices are serviced at most *once*. For example, if the priority and "in service" status is:

Before Rotate (IR4 the highest priority requiring service)

	IS7	IS6	IS5	IS4	IS3	IS2	IS1	IS0
"IS" Status	0	1	0	1	0	0	0	0

	Lowest Priority							Highest Priority
Priority Status	7	6	5	4	3	2	1	0

After Rotate (IR4 was serviced, all other priorities rotated correspondingly)

	IS7	IS6	IS5	IS4	IS3	IS2	IS1	IS0
"IS" Status	0	1	0	0	0	0	0	0

	Highest Priority							Lowest Priority
Priority Status	2	1	0	7	6	5	4	3

There are two ways to accomplish Automatic Rotation using OCW2, the Rotation on Non-Specific EOI Command (R = 1, SL = 0, EOI = 1) and the Rotate in Automatic EOI Mode which is set by (R = 1, SL = 0, EOI = 0) and cleared by (R = 0, SL = 0, EOI = 0).

SPECIFIC ROTATION (Specific Priority)

The programmer can change priorities by programming the bottom priority and thus fixing all other priorities; i.e., if IR5 is programmed as the bottom priority device, then IR6 will have the highest one.

The Set Priority command is issued in OCW2 where: R = 1, SL = 1; LO–L2 is the binary priority level code of the bottom priority device.

Observe that in this mode internal status is updated by software control during OCW2. However, it is independent of the End of Interrupt (EOI) command (also executed by OCW2). Priority changes can be executed during an EOI command by using the Rotate on Specific EOI command in OCW2 (R = 1, SL = 1, EOI = 1 and LO–L2 = IR level to receive bottom priority).

INTERRUPT MASKS

Each Interrupt Request input can be masked individually by the Interrupt Mask Register (IMR) programmed through OCW1. Each bit in the IMR masks one interrupt channel if it is set (1). Bit 0 masks IR0, Bit 1 masks IR1 and so forth. Masking an IR channel does not affect the other channels operation.

AFN-00221E

routine to dynamically alter the system priority structure during its execution under software control. For example, the routine may wish to inhibit lower priority requests for a portion of its execution but enable some of them for another portion.

The difficulty here is that if an Interrupt Request is acknowledged and an End of Interrupt command did not reset its IS bit (i.e., while executing a service routine), the 8259A would have inhibited all lower priority requests with no easy way for the routine to enable them.

That is where the Special Mask Mode comes in. In the special Mask Mode, when a mask bit is set in OCW1, it inhibits further interrupts at that level *and enables* interrupts from *all other* levels (lower as well as higher) that are not masked.

Thus, any interrupts may be selectively enabled by loading the mask register.

The special Mask Mode is set by OCW3 where: SSMM = 1, SMM = 1, and cleared where SSMM = 1, SMM = 0.

processor internal Interrupt Enable flip-flop is reset, disabling its interrupt input. Service to devices is achieved by software using a Poll command.

The Poll command is issued by setting P = "1" in OCW3.

The 8259A treats the next $\overline{RD}$ pulse to the 8259A (i.e., $\overline{RD} = 0$, $\overline{CS} = 0$) as an interrupt acknowledge, sets the appropriate IS bit if there is a request, and reads the priority level. Interrupt is frozen from $\overline{WR}$ to $\overline{RD}$.

The word enabled onto the data bus during $\overline{RD}$ is:

D7	D6	D5	D4	D3	D2	D1	D0
I	—	—	—	—	W2	W1	W0

W0-W2: Binary code of the highest priority level requesting service.

I: Equal to a "1" if there is an interrupt.

This mode is useful if there is a routine command common to several levels so that the $\overline{INTA}$ sequence is not needed (saves ROM space). Another application is to use the poll mode to expand the number of priority levels to more than 64.

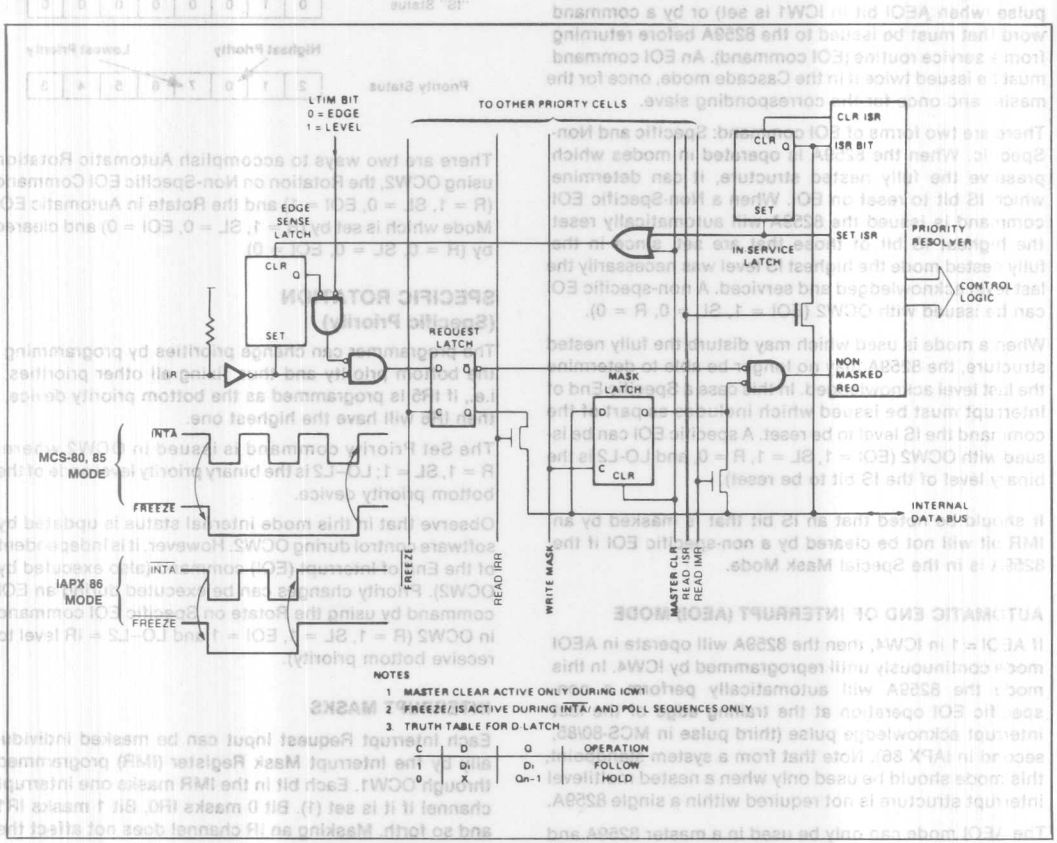


Figure 9. Priority Cell—Simplified Logic Diagram

AFN 00221E

READING THE 8259A STATUS

The input status of several internal registers can be read to update the user information on the system. The following registers can be read via OCW3 (IRR and ISR or OCW1 [IMR]).

Interrupt Request Register (IRR): 8-bit register which contains the levels requesting an interrupt to be acknowledged. The highest request level is reset from the IRR when an interrupt is acknowledged. (Not affected by IMR.)

In-Service Register (ISR): 8-bit register which contains the priority levels that are being serviced. The ISR is updated when an End of Interrupt Command is issued.

Interrupt Mask Register: 8-bit register which contains the interrupt request lines which are masked.

The IRR can be read when, prior to the RD pulse, a Read Register Command is issued with OCW3 (RR = 1, RIS = 0.)

The ISR can be read when, prior to the RD pulse, a Read Register Command is issued with OCW3 (RR = 1, RIS = 1).

There is no need to write an OCW3 before every status read operation, as long as the status read corresponds with the previous one; i.e., the 8259A "remembers" whether the IRR or ISR has been previously selected by the OCW3. This is not true when poll is used.

After initialization the 8259A is set to IRR.

For reading the IMR, no OCW3 is needed. The output data bus will contain the IMR whenever $\overline{RD}$ is active and AO =1 (OCW1).

Polling overrides status read when P = 1, RR = 1 in OCW3.

EDGE AND LEVEL TRIGGERED MODES

This mode is programmed using bit 3 in ICW1.

If LTIM = '0', an interrupt request will be recognized by a low to high transition on an IR input. The IR input can remain high without generating another interrupt.

If LTIM = '1', an interrupt request will be recognized by a 'high' level on IR Input, and there is no need for an edge detection. The interrupt request must be removed before the EOI command is issued or the CPU interrupt is enabled to prevent a second interrupt from occurring.

The priority cell diagram shows a conceptual circuit of the level sensitive and edge sensitive input circuitry of the 8259A. Be sure to note that the request latch is a transparent D type latch.

In both the edge and level triggered modes the IR inputs must remain high until after the falling edge of the first INTA. If the IR input goes low before this time a DEFAULT IR7 will occur when the CPU acknowledges the interrupt. This can be a useful safeguard for detecting interrupts caused by spurious noise glitches on the IR inputs. To implement this feature the IR7 routine is used for "clean up" simply executing a return instruction, thus ignoring the interrupt. If IR7 is needed for other purposes a default IR7 can still be detected by reading the ISR. A normal IR7 interrupt will set the corresponding ISR bit, a default IR7 won't. If a default IR7 routine occurs during a normal IR7 routine, however, the ISR will remain set. In this case it is necessary to keep track of whether or not the IR7 routine was previously entered. If another IR7 occurs it is a default.

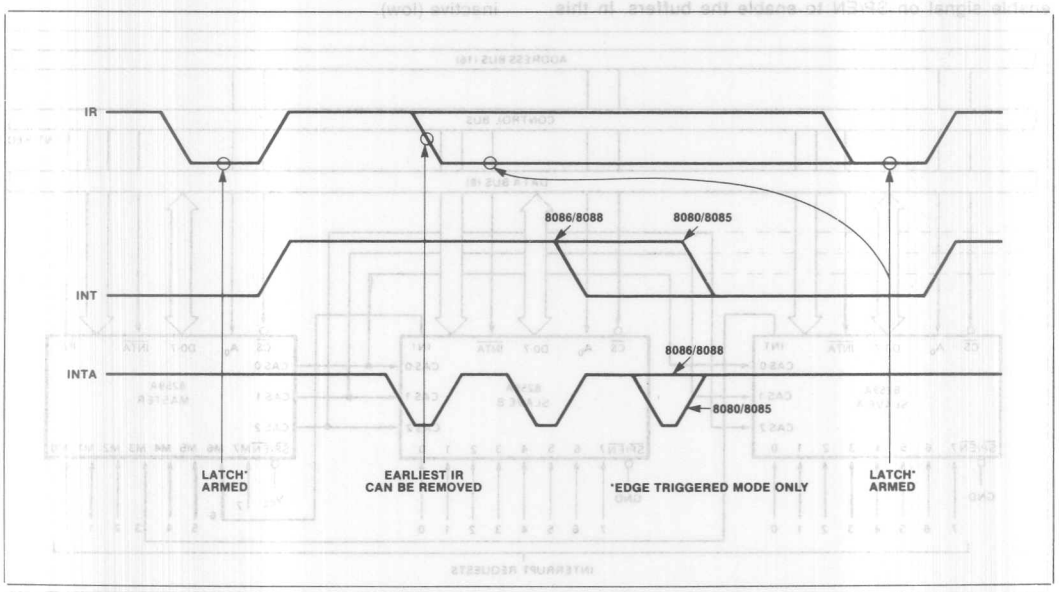


Figure 10. IR Triggering Timing Requirements

AFN-00221E

THE SPECIAL FULLY NESTED MODE

This mode will be used in the case of a big system where cascading is used, and the priority has to be conserved within each slave. In this case the fully nested mode will be programmed to the master (using ICW4). This mode is similar to the normal nested mode with the following exceptions:

a. When an interrupt request from a certain slave is in service this slave is not locked out from the master's priority logic and further interrupt requests from higher priority IR's within the slave will be recognized by the master and will initiate interrupts to the processor. (In the normal nested mode a slave is masked out when its request is in service and no higher requests from the same slave can be serviced.)

b. When exiting the Interrupt Service routine the software has to check whether the interrupt serviced was the only one from that slave. This is done by sending a non-specific End of Interrupt (EOI) command to the slave and then reading its In-Service register and checking for zero. If it is empty, a non-specific EOI can be sent to the master too. If not, no EOI should be sent.

BUFFERED MODE

When the 8259A is used in a large system where bus driving buffers are required on the data bus and the cascading mode is used, there exists the problem of enabling buffers.

The buffered mode will structure the 8259A to send an enable signal on $\overline{SP}/\overline{EN}$ to enable the buffers. In this mode, whenever the 8259A's data bus outputs are enabled, the $\overline{SP}/\overline{EN}$ output becomes active.

This modification forces the use of software programming to determine whether the 8259A is a master or a slave. Bit 3 in ICW4 programs the buffered mode, and bit 2 in ICW4 determines whether it is a master or a slave.

CASCADE MODE

The 8259A can be easily interconnected in a system of one master with up to eight slaves to handle up to 64 priority levels.

The master controls the slaves through the 3 line cascade bus. The cascade bus acts like chip selects to the slaves during the $\overline{INTA}$ sequence.

In a cascade configuration, the slave interrupt outputs are connected to the master interrupt request inputs. When a slave request line is activated and afterwards acknowledged, the master will enable the corresponding slave to release the device routine address during bytes 2 and 3 of INTA. (Byte 2 only for 8086/8088).

The cascade bus lines are normally low and will contain the slave address code from the trailing edge of the first INTA pulse to the trailing edge of the third pulse. Each 8259A in the system must follow a separate initialization sequence and can be programmed to work in a different mode. An EOI command must be issued twice: once for the master and once for the corresponding slave. An address decoder is required to activate the Chip Select (CS) input of each 8259A.

The cascade lines of the Master 8259A are activated only for slave inputs, non slave inputs leave the cascade line inactive (low).

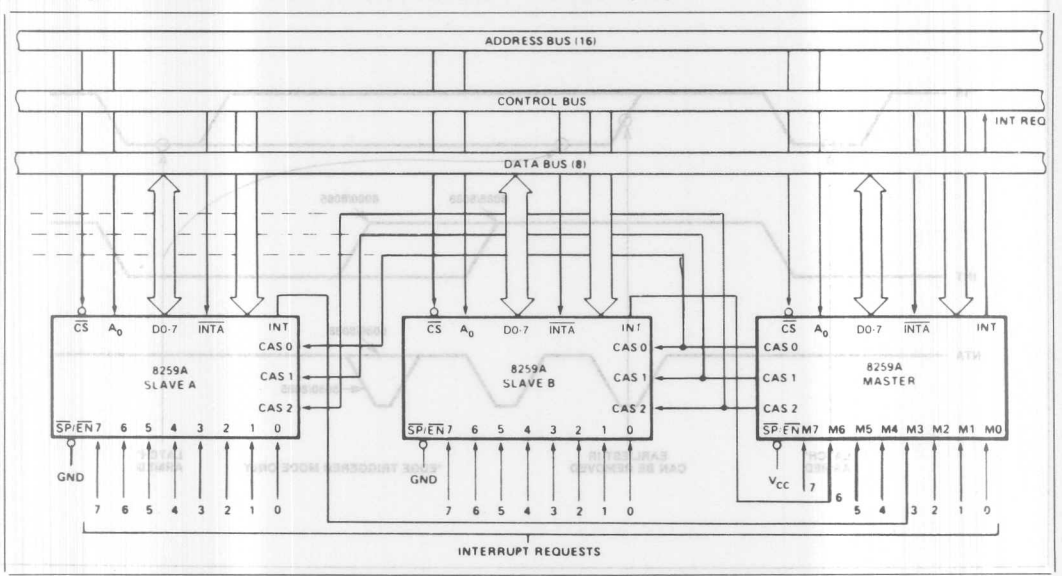


Figure 11. Cascading the 8259A

AFN 00221E

ABSOLUTE MAXIMUM RATINGS*

Ambient Temperature Under Bias 0°C to 70°C
Storage Temperature −65°C to +150°C
Voltage on Any Pin
with Respect to Ground −0.5V to +7V
Power Dissipation 1 Watt

NOTICE: Stresses above those listed under "Absolute Maximum Ratings" may cause permanent damage to the device. This is a stress rating only and functional operation of the device at these or any other conditions above those indicated in the operational sections of this specification is not implied.

D.C. CHARACTERISTICS [T_A = 0°C to 70°C, V_{CC} = 5V ±5% (8259A-8), V_{CC} = 5V ±10% (8259A, 8259A-2)]

Symbol	Parameter	Min.	Max.	Units	Test Conditions
V_{IL}	Input Low Voltage	−0.5	0.8	V	
V_{IH}	Input High Voltage	2.0*	V_{CC} +0.5V	V	
V_{OL}	Output High Voltage		0.45	V	I_{OL} = 2.2mA
V_{OH}	Output High Voltage	2.4		V	I_{OH} = −400μA
$V_{OH(INT)}$	Interrupt Output High Voltage	3.5		V	I_{OH} = −100μA
		2.4		V	I_{OH} = −400μA
I_{LI}	Input Load Current	−10	+10	μA	$0V \leq V_{IN} \leq V_{CC}$
I_{LOL}	Output Leakage Current	−10	+10	μA	$0.45V \leq V_{OUT} \leq V_{CC}$
I_{CC}	V_{CC} Supply Current		85	mA	
I_{LIR}	IR Input Load Current		−300	μA	V_{IN} = 0
			10	μA	$V_{IN} = V_{CC}$

*Note: For Extended Temperature EXPRESS V_{IH} = 2.3V.

CAPACITANCE (T_A = 25°C; V_{CC} = GND = 0V)

Symbol	Parameter	Min.	Typ.	Max.	Unit	Test Conditions
C_{IN}	Input Capacitance			10	pF	fc = 1 MHZ
$C_{I/O}$	I/O Capacitance			20	pF	Unmeasured pins returned to V_{SS}

A.C. CHARACTERISTICS [T_A = 0°C to 70°C, V_{CC} = 5V ±5% (8259A-8), V_{CC} = 5V ± 10% (8259A, 8259A-2)]

TIMING REQUIREMENTS

Symbol	Parameter	8259A-8		8259A		8259A-2		Units	Test Conditions
		Min.	Max.	Min.	Max.	Min.	Max.		
TAHRL	AO/$\overline{CS}$ Setup to $\overline{RD}$/$\overline{INTA}$↓	50		0		0		ns	
TRHAX	AO/$\overline{CS}$ Hold after $\overline{RD}$/$\overline{INTA}$↑	5		0		0		ns	
TRLRH	$\overline{RD}$ Pulse Width	420		235		160		ns	
TAHWL	AO/$\overline{CS}$ Setup to $\overline{WR}$↓	50		0		0		ns	
TWHAX	AO/$\overline{CS}$ Hold after $\overline{WR}$↑	20		0		0		ns	
TWLWH	$\overline{WR}$ Pulse Width	400		290		190		ns	
TDVWH	Data Setup to $\overline{WR}$↑	300		240		160		ns	
TWHDX	Data Hold after $\overline{WR}$↑	40		0		0		ns	
TJLJH	Interrupt Request Width (Low)	100		100		100		ns	See Note 1
TCVIAL	Cascade Setup to Second or Third $\overline{INTA}$↓ (Slave Only)	55		55		40		ns	
TRHRL	End of $\overline{RD}$ to next $\overline{RD}$ End of $\overline{INTA}$ to next $\overline{INTA}$ within an $\overline{INTA}$ sequence only	160		160		160		ns	
TWHWL	End of $\overline{WR}$ to next $\overline{WR}$	190		190		190		ns	

AFN-00221E

A.C. CHARACTERISTICS (Continued)

Symbol	Parameter	8259A-8		8259A		8259A-2		Units	Test Conditions
		Min.	Max.	Min.	Max.	Min.	Max.		
*TCHCL	End of Command to next Command (Not same command type)	500		500		500		ns	
	End of $\overline{INTA}$ sequence to next $\overline{INTA}$ sequence.								

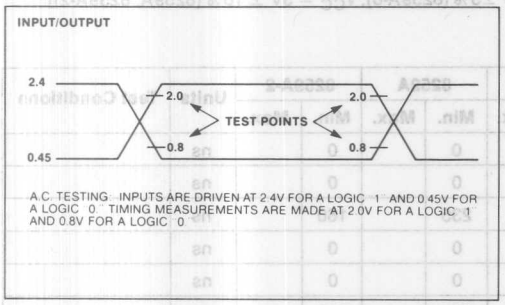
*Worst case timing for TCHCL in an actual microprocessor system is typically much greater than 500 ns (i.e. 8085A = 1.6μs, 8085A-2 = 1μs, 8086 = 1μs, 8086-2 = 625 ns)

NOTE: This is the low time required to clear the input latch in the edge triggered mode.

TIMING RESPONSES

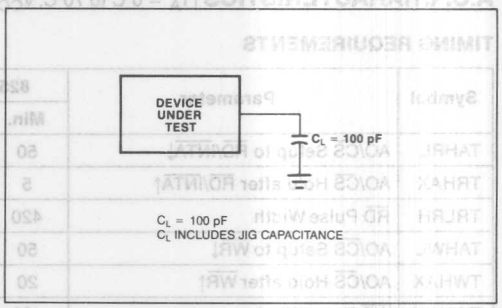
Symbol	Parameter	8259A-8		8259A		8259A-2		Units	Test Conditions
		Min.	Max.	Min.	Max.	Min.	Max.		
TRLDV	Data Valid from $\overline{RD}$/$\overline{INTA}$↓		300		200		120	ns	C of Data Bus = 100 pF
TRHDZ	Data Float after $\overline{RD}$/$\overline{INTA}$↑	10	200	10	100	10	85	ns	C of Data Bus Max text C = 100 pF Min. test C = 15 pF
TJHIH	Interrupt Output Delay		400		350		300	ns	
TIALCV	Cascade Valid from First $\overline{INTA}$↓ (Master Only)		565		565		360	ns	C_{INT} = 100 pF
TRLEL	Enable Active from $\overline{RD}$↓ or $\overline{INTA}$↓		160		125		100	ns	$C_{CASCADE}$ = 100 pF
TRHEH	Enable Inactive from $\overline{RD}$↑ or $\overline{INTA}$↑		325		150		150	ns	
TAHDV	Data Valid from Stable Address		350		200		200	ns	
TCVDV	Cascade Valid to Valid Data		300		300		200	ns	

A.C. TESTING INPUT, OUTPUT WAVEFORM



A.C. TESTING: INPUTS ARE DRIVEN AT 2.4V FOR A LOGIC "1" AND 0.45V FOR A LOGIC "0." TIMING MEASUREMENTS ARE MADE AT 2.0V FOR A LOGIC "1" AND 0.8V FOR A LOGIC "0."

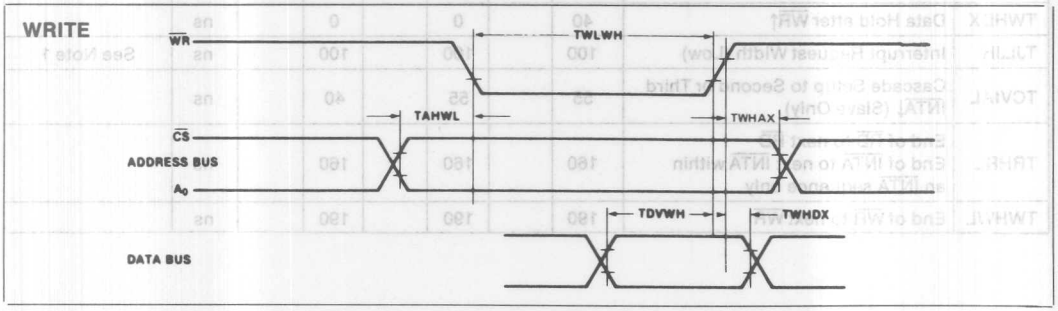
A.C. TESTING LOAD CIRCUIT



C_L = 100 pF
C_L INCLUDES JIG CAPACITANCE

WAVEFORMS

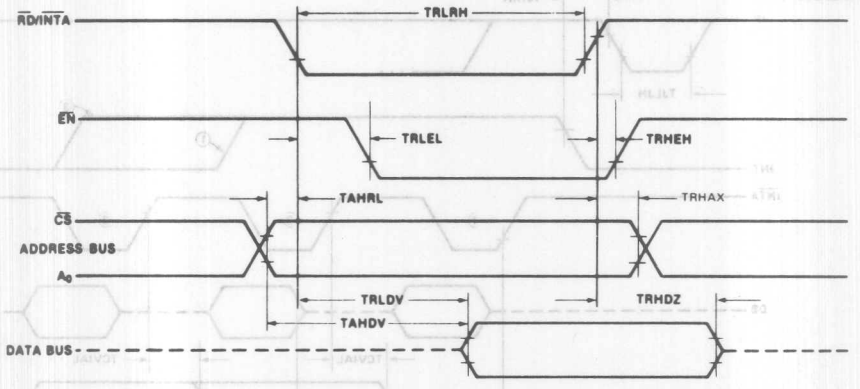
WRITE



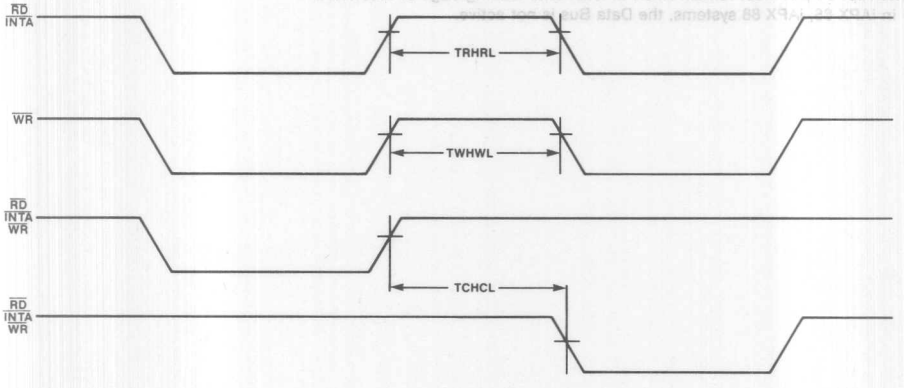
AFN 00221E

WAVEFORMS (Continued)

READ/INTA



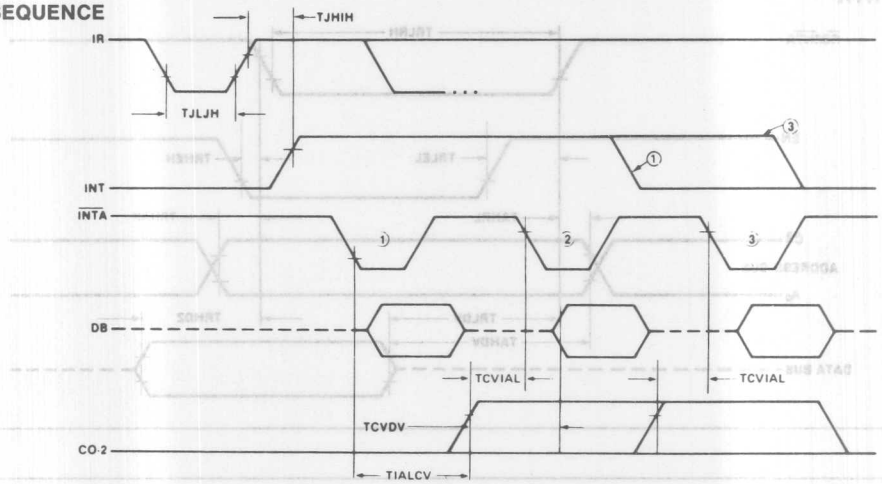
OTHER TIMING



AFN 00221E

WAVEFORMS (Continued)

INTA SEQUENCE



NOTES: Interrupt output must remain HIGH at least until leading edge of first INTA.
1. Cycle 1 in iAPX 86, iAPX 88 systems, the Data Bus is not active.

AFN 00221E